UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

MASTER THESIS

Data Semantic Enrichment for Complex Event Processing over IoT Data Streams

Author

Patrick Schneider

Date: February, 2019

Supervisor

Prof. Fatos Xhafa, Department of Computer Science

Declaration of originality

I, **Patrick Schneider** , hereby confirm that I am the sole author of the thesis with the title **Data Semantic Enrichment for Complex Event Processing over IoT Data Streams** and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Data:
15.10.2019

Signature:

Abstract

With the ever-increasing number of connected devices, the IoT devices have become a primary source of data generation in the form of Big Data or Big Data streams. Current research is investigating how to process data along the data pipeline, use resources available outside the cloud, and semantically enrich data streams for unified representations and complex event processing inferences.

This thesis generalizes techniques for processing an IoT data stream, semantically enrich the data with contextual information at Edge, Fog, or Cloud-layer, as well as complex event processing in IoT applications.

A case study in the field of eHealth is used to validate the knowledge foundation of this thesis. The use case demonstrates an anomaly detection, and classification scenario over an *Electrocardiogram* (ECG) stream, where anomalous ECG signals are processed dynamically across the data pipeline and classified with modern machine learning approaches based on the *Hierarchical temporal memory* (HTM) and *Convolutional Neural Network* (CNN) algorithms. By applying the HTM algorithm on the Edge layer, the data volume can be reduced over the IoT ecosystem, which results in the reduction of classification latency as well as needed processing resources. The proposed architecture of the case study describes an adaptive solution that can be extended to other use cases to enable complex analysis of patient data in a historical, predictive, and even prescriptive application scenario. The resulting implementation acts as a proposal for researchers and developers to rapidly deploy and test scalable IoT scenarios in the field of data processing, data enrichment, and (semantic-) complex event processing.

Acknowledgement

I want to thank my thesis advisor Prof. Fatos Xhafa of the Computer Science Department of the Technical University of Catalonia (UPC). The door to Prof. Xhafa's office was always open whenever I had a question about my research or writing. He allowed me to conduct my research by steering me in the right direction. Thank you!

Further, I want to thank my parents, Eva and Heinrich, as well as my twin brother, Steffen, for providing me with unconditional love and support in my life decisions. Thank you to Kaoutar, for all the love and happiness you give me.

Patrick Schneider

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API *Application Programming Interface*

BASIL *BuildingAPIs SImpLy*

BGP *Basic Graph Pattern*

BN *Bayesian Networks*

CEP *Complex Event Processing*

CNN *Convolutional Neural Network*

CQL *Continuous Query Language*

CQP *Continuous Query Processing*

CVD *Cardiovascular disease*

DBMS *Database Management System*

DFT *Discrete Fourier Transformation*

DL *Description Logic*

DSMS *Data Stream Management System*

DTW *Dynamic Time Warping*

ECA *Event-Condition-Actions*

ECG *Electrocardiogram*

EPA *Event Processing Agent*

EPL *Event Processing Language*

FCN *Fog Computing Nodes*

FS *Fast Shapelets*

HMM *Hidden Markov Models*

HTM *Hierarchical temporal memory*

ICU *Intensive Care Unit*

IDG *International Data Group*

IoT *Internet of Things*

KAT *Knowledge Acquisition Toolkit*

LER *Linked Edit Rules*

LS *Learned Shapelets*

LOF *Local Outlier Factor*

LOR *Linked Open Rules*

LOV *Linked Open Vocabulary*

LS *Learned Shapelets*

MEC *Mobile Edge Computing*

MACC *Mobile ad-hoc Cloud Computing*

MCC *Mobile Cloud Computing*

MOM *Message oriented Middleware*

PLR *Piecewise Linear Representation*

RDF *Resource Description Framework*

S-LOR *Sensor-based Linked Open Rules*

SAX *Symbolic Aggregate Approximation*

SCEP *Semantic Complex Event Processing*

SSN *Semantic Sensor Network*

ST *Shapelet Transform*

SVM *Support Vector Machine*

UDA *User-Defined Aggregates*

W3C *World Wide Web Consortium*

WoT *Web of Thing*

Chapter 1

Introduction

This section contains the introduction to the *Internet of Things* (IoT) streaming domain and the motivation of this thesis. Further, the challenges of modern IoT streaming applications are elaborated. Based on those challenges, the problem statement and goals of the thesis are defined. Concluding this section, the structure of this work are outlined.

1.1 Motivation

IoT-capable devices can produce massive amounts of data. Cisco predicts that by 2030, 500 billion devices will be expected to be connected to the internet [C16]. Smart devices generate data for IoT applications to aggregate, analyze, and create insight, to drive informed decisions and actions. An essential component of smart devices are sensors that gather information about the environment in which they are deployed. Many sensors can be combined to create an application that fulfills a global purpose. For instance, an autonomously driving car generates a data volume of approximately 4 terabytes per day [Nis19]. Airplanes create an even higher data volume. The wing of an Airbus A380, contains around 10,000 sensors, where each wing delivers around 5 terabytes of data per day. The sensors within the engines alone measure around 5,000 parameters 16 times per second. Just the individual information signals add up to around 150 million throughout a flight that need to be analyzed [DC18]. However, the majority of this information is disposable data, characterized by minimal or even missing potential for later reuse. While the given examples are prominent, do the same characteristics hold to smaller IoT sensor applications. Depending on the use case, IoT applications need the analysis in real-time for immediate decision-making.

Edge computing is an indispensable key technology for the IoT. In this paradigm, data is processed in a decentralized way, close to where the data is created. International Data Group *International Data Group* (IDG) analysts predicted that by 2019, already 43 percent of the IoT generated data would be processed on the edge of the network with edge computing systems to handle the flood of data [MTL+16]. Edge computing shifts computing to the edge of the network to further guarantee low latency and prevent bottlenecks in data delivery or reasoning. Furthermore, Edge computing helps to ease the burden on conventional central computing architectures and helps to protect mission-critical data and services by reducing the need for central processing [KNL+18].

Devices as small as Raspberry Pis can handle data processing for a variety of IoT endpoints. However, their performance is barely scalable, and their availability is hardly guaranteed.

Industries, health care, and cities are exploiting IoT data-driven frameworks to make their environment more efficient and effective. For making IoT a reality, data produced by sensors, smartphones, watches, and other wearables need to be integrated and the meaning of IoT data extracted. To extract meaning from heterogeneous devices and data environments, the data structure needs to be defined in a unified representation schema and semantically linked.

1.2 Problem Statement and Research Questions

With the paradigm of IoT real-time analytics, the challenges for this thesis are described in the 4 main categories:

1. Variety, Velocity and data rate of IoT data Streams

IoT Devices generate continuous data at a very high rate that needs to be collected and structured. Videos, tweets, audio recordings, documents, or ECG strips are all data but are generally unstructured, and varied. Velocity is the measure of how fast the data is arriving in IoT infrastructure components, where the goal is to reduce the data rate as much as possible without losing essential events and information.

2. Efficiency - Timeliness and Instantaneity Efficiency in IoT streaming systems can be considered in the aspect of low latency and reliable delivery on limited resources.

Low latency is required in critical applications or complex situations where the data needs to be processed before it gets outdated. On resource-constrained IoT devices, a high efficiency enables processing close to the data source. This reduces communication cost, secures better data availability, faster event detection, and reasoning, with what IoT cloud systems can be spared by having a decreased data rate.

3. Robustness - Randomness and Imperfection IoT streamed data often has the characteristic of being incomplete and unordered in their arrival time and unreliable. This can lead to different challenges in the robustness of IoT services. Precise reasoning can hardly be achieved because IoT stream data is observed through a (time-) window of a specific size. In some applications, the whole input can not be considered in the processing, which can lead to an incomplete inference. Building suitable models for uncertainty management is an important consideration in several IoT stream processing applications.

4. Semantic Expressive Power While the main challenges of IoT data communication at sensor level are being solved, handling the heterogeneity of data on a semantic level is still a major issue [Hud16]. IoT sensor devices produce a low level of data. To understand complex situations, the challenge lies in extracting higher knowledge levels on a low heterogeneous data level. For an efficient knowledge deduction on IoT data, knowledge extraction and processing models are required. Those models build the basis for semantic integration of heterogeneous sources. Heterogeneous data further contains the challenge of transforming various data formats, which differ in their semantic expressive power.

Research Question Based on the aforementioned challenges, the focus on this thesis can be formulated in the following research questions:

1. What primitive functions can be used to process IoT streaming data efficiently? This question builds the basis of processing IoT data streams on the lowest level close to the source and along the data pipeline. In comparison to traditional *Database Management System* (DBMS), the nature of continuous data is subject to processing limitations and new paradigms. The focus will be held on identifying the fundamental functions of IoT data stream processing as well as classifying more sophisticated methods that are used depending on the type of IoT data stream present.

2. What operations need to be defined to reason over complex events? This question builds on the results of the previous research question and studies the follow-up step of how the initially processed data and detected primitive events could be used as a basis to infer knowledge and reason over more complex event scenarios.

3. What semantic operations need to be defined to enrich data with context information to link events semantically? This question concerns the challenge of standardizing semantic expressive power in homogeneous and heterogenous IoT application environment. The focus lies on finding best-practice approaches or standards to semantically enrich information uniformly and reviews methodologies in the field of knowledge base integrations in the field of IoT Data.

1.3 Case Study: Scalable IoT patient data processing and reasoning ecosystem in the field of health analysis

A good health monitoring system will discover abnormalities of health conditions in time and build diagnoses in line with the inferred knowledge. The most important approach to diagnosis is sensor-based patient monitoring. However, in the past, this monitoring was unlikely to be transportable. As these devices are typically too costly for home use, patients need to move in many cases to hospitals where they were restricted during the time of information collection. The burden of hospitals in terms of space requirements, cost, staffing, and possible advanced analytics results increases the need for modern patient monitoring approaches.

The objective of this selected case study is to validate and apply the concepts and methodologies explained in this thesis, where stream processing, semantic enrichment, and complex event processing showcases a proposal that holds on the main requirements for a health-data real-time analytics ecosystem. The proposed architecture evaluates the use case of processing ECG data in real-time, starting from the sensor, over the Edge/Fog-Layer to the central analytics server. ECG data can be considered as real-time IoT data streams, in which the processing can be critical. The idea behind the ECG analysis is to preprocess the raw data and check the signals for anomalies so that the successive processing load can be reduced, as well as the analytics focused on relevant data. Relevant elements should be ingested and analyzed for primitive events

in the Edge or Fog layer. With the possibility of retrieving primitive events, the proof of concept for *Complex Event Processing* (CEP) will be showcased in a possible integration in the defined ecosystem.

The case study includes the following milestones:

1. Design of an IoT real-time ecosystem for semantic stream analytics functionality.
2. Design of a data processing pipeline for ECG analysis and CEP integration, while the processing pipeline should be generalized on other use cases in the field of IoT streams.
3. Implementation of anomaly detection, primitive event classification, and CEP to validate the proof of concept of the designed real-time analytics ecosystem.
4. Description of possible semantic complex event processing and knowledge base integration.

1.4 Thesis structure

Chapter 2 "Related Work" describes the state of the art and other findings inside the research communities in the field of IoT healthcare applications and semantic reasoning over IoT data streams. Chapter 3 "Overview" builds the theoretical foundation of the domains of IoT stream processing, enrichment, and reasoning. An introduction of modern IoT architecture designs will be introduced and elaborated in chapter 4 "IoT Edge Architecture", in the context of implementation choices based on use case characteristics. The elaboration of the case study follows in chapter 5 "Scalable IoT patient data processing and reasoning ecosystem in the field of health analysis", where the previous chapters serve as decision foundation over the chosen implementation of this proof of concept. The end of this thesis represents chapter 6 "Conclusion and future work", which critically assess the results of the case study, as well as gives insights into current open research questions in the explored topics of semantic enrichment and complex event processing over IoT edge streaming. The future work will give an outlook on possible improvements to the case study.

Chapter 2

Related work and literature review

Big IoT Data for Remote Health Monitoring

[IKK+15] showed an overview of existing IoT–based healthcare architectures, platforms, and industry trends. Also, they presented an extensive overview of IoT healthcare systems, one of them being the Internet of m-Health Things (m-IoT), which represents a consolidation of IoT sensors, mobile computing, and communications technologies. [MNG+17] focused on the data analytics aspect of IoT architecture, opportunities, and challenges. A brief overview of research efforts directed toward IoT data analytics between IoT and Big Data was given. Another evaluation was conducted on analytic types, methods, and technologies in the field of IoT data mining. [MGT+17] proposed a distributed IoT framework in monitoring activities involving physical exertion over biomedical signals. [MWY+17] presented an IoT based health application system, leveraging big data and IoT. The architecture acts as a basis to described challenges and potential m-health applications. The smart health concept, for the integration of health principles with sensors and information in smart cities, was introduced by [SPC+14]. The paper provided an overview of smart health principles and the main challenges and opportunities of smart healthcare in the context of smart city concepts.

Processing and Analysis of health data streams

[FPY+16] has given an overview of challenges and techniques for data processing of big data in health informatics. Machine learning algorithms were characterized and compared. This paper resulted in a proposal for a general processing pipeline for healthcare data that includes data sensing, storing, analyzing, search, and discover, as well as decision support.

Processing and Analytics on the Edge and Cloud layer

[SHT16] presented a comparative analysis of state of the art processing solutions over Open-Source solutions and commercial stream solutions. The comparison characteristics where the processing model, latency, data pipeline, fault tolerance, and reliability.

Findings and Contribution:

- The utilization of biomedical sensing devices is growing [sources].
- Advanced application use cases demand the need for offloading of computing and analytics tasks, due to the resource limitations on IoT devices.
- The increasing number of connected IoT devices and new types of applications enables new interconnected use case scenarios.

In current scientific work, use cases are often illustrated in the optimization of data processing and reasoning, while the fields of semantic interoperability still represent a challenge, depending on the domain of application. Based on this observation contains this work, the contribution of:

- **Contribution 1**: Giving a use case independent foundation over the fields of stream processing, semantic enrichment, and complex event processing.
- **Contribution 2**: Development of a use case based prototype that enables an end-to-end data pipeline process with the requirement for a generalized and scalable IoT stream infrastructure in the fields of stream processing, semantic enrichment, and complex event processing.

Chapter 3

Overview

In this chapter, the processing pipeline of IoT data is introduced, starting with the main characteristics of stream data in section 3.1. It follows the introduction in section 3.2, that describes how the stream data can be processed and which reasoning techniques exist for IoT data streams in simple and complex event scenario. The principles and methods for semantic IoT data will be presented in section 3.3).

3.1 IoT Streams in the context of big data

The core utility of data stream analytics is the recognition and extraction of meaningful patterns from raw data inputs. With that, a higher level of insight can be retrieved and used for event detection, complex event processing, reasoning, and decision making. Extracting knowledge from raw data is essential for many applications and businesses since it potentially enables competitive advantages. Several works have characterized the characteristics of big data from different points in terms of volume, velocity, and variety. A commonly used model today is the ”6V”, that is used to characterize IoT Big Data, where also data streams can be described with:

Volume represents the amount of data generated. The estimated data generation of an autonomous car lays at approximately 300TB per year [aut].

Velocity is the speed in which new data is generated and moved trough networks. It is a crucial characteristic to consider especially for critical tasks like credit card transaction checks for fraudulent activities or the sensors of an autonomous car.

Variety represents the different types of data representation, where the differentiation not only lies in the general type, whether the streaming data is text or numbers. Based on different sensor manufacturers, the same sensor application can have different data schemes.

Veracity represents the inaccuracy as well as the trustworthiness of data. Data streams appear in many forms where quality and accuracy are hard to manage. The volumes often balance the lack of quality or accuracy, where the aspects of real-time analytics stay in a trade-off between volume-driven accuracy and fast analytics.

Value results from the transformation of raw data to meaningful insights. The Value highly depends on the underlying processes, services, and the way the data is processed. For example, an ECG signal monitoring may need to sense on all sensor data, opposed to a weather forecast sensor only needs random samples of sensor data. In the context of real-time analytics, the value of the data could decay, the slower it is processed.

Variability represents the different data flow rates. Depending on the field in which IoT devices are applied, inconsistent data flows can result. A data source can have different data load rates based on specific times. For example, in environmental monitoring, devices are required to maximize their battery life. This results in a need for efficient processing and low network overhead. The processing speed, however, is usually not as crucial in these cases. In a production environment, the power usage of the sensing devices and processing is not a concern, as power is readily available, and the usage is marginal compared to the production line. The data rates in these kinds of environments can be very high, and there may be a need for instantaneous processing. An error in the production line that is not detected within seconds or even milliseconds may result in production issues.

Batch vs. Realtime analytics: The streaming data analytics on high-performance computing systems or cloud platforms is mainly based on data parallelism and incremental processing [LDS15]. By data parallelism, a large dataset is partitioned into several subsets, on which parallel analytics can be performed simultaneously. Incremental or serial processing refers to fetching a small sample of batch data to be processed quickly in a pipeline. Even though these techniques reduce the response latency

from the streaming analytics framework, they might be not the best solution for stream IoT applications. By bringing stream analytics closer to the source of IoT and edge devices, the need for central data parallelism and sequential processing is less sensible.

3.1.1 Static Data - DBMS vs Continuous Data - DSMS Systems

The main objective of both *Data Stream Management System* (DSMS)s and DBMSs is to provide generic data management for applications. Still, there are differences how to manage data and evaluate queries. DSMSs have their origin in DBMSs but present substantial differences.The following paragraphs will evaluate the different characteristics.

Query Types The first difference can be seen in the type of queries, where DBMS run one-time queries over persistently stored data, while DSMS system make use of continuous queries over transient data. A DBMS query is executed and gives the output for the current state of the relations. DSMS on the other side are long-running queries and stay active in the system over a longer time window. Once a DSMS registered a continuous query, the results will be generated as output continuously over new arriving stream elements until it is deregistered.

Query Answers The query answer of a DBMS always produce exact answers for a given query. Continuous queries, on the other side, usually provide approximate query answers. The reasons for this approximate answers are that

(i) many continuous queries are not computable with a finite amount of memory. An example of this is the cartesian product over two infinite streams, which lead in bounded computation to an approximate answer.

(ii) The group-by operator would block an infinite stream for exact answers, which is why an approximate result over a finite space is giving.

(iii) The accumulation of data can be faster than the system can process. In the context of continuous streaming, high quality approximated data is accepted as an answer. Moreover, newly arrived data is seen as more accurate and relevant in continuous data than in old data.

Processing Methodology DBMS can be seen as demand-driven computation models, where the processing of a query is issued. The tuples are read from the persistently

stored data via scan or index-based access methods. *Continuous Query Processing* (CQP) in a DSMS is a data-driven approach, where the answer is computed incrementally on the arrival of new stream elements. Without the use of buffering, DSMS can access the stream elements only in a sequential arriving order, where DBMS can access tuples randomly.

Query Optimization The query optimization in DBMS is done before the execution. The optimizer generates semantically equivalent query executions plans, based on cost approximation, which contains different performance characteristics. The approximation of the cost is handled with statistical information of the tables, relations, and system information. The optimizer chooses the plan afterward with the lowest estimated cost. Continuous queries in DSMS require query optimization and run-time to adapt to changing stream characteristics and system conditions. Query workload can change over time, as well as data distributions and arrival rates. To not degrade the performance of the stream application, run-time optimization is an important aspect to consider in DSMS.

In the rest of this thesis, the focus will be held singly on the CQP paradigm.

3.1.2 Time variability - Characterizing continuous data streams

In continuous data streams, the validity and usefulness of stream elements have a lifespan. Latency in a network or out-of-date stream elements could degrade the value of the analysis. Also is the anticipated time variability of the data streams necessary for later processing steps and reasoning. An example can be that an alarm will be triggered over an average threshold of the last 10 data elements. The problem occurs if the time variability is not constant, and only 9 data elements arrived, and further elements are not generated in a constant time interval.

In this thesis, the data stream will be generally described as:

1. A data stream is a possible unbounded sequence of data items generated by a data source. A single data item is called a stream element.

2. Stream elements usually arrive continuously at a system.

3. The core system has no control over the sequence of the arriving elements as well as their arrival rates. The stream rates and ordering can be unpredictable and change over time.

Figure 3.1: Illustration of data flow with fixed time intervals t1

4. A data source sends the stream elements only once. As stream elements are accessed sequentially, a past arrived stream element cannot be retrieved unless it is stored. With the unbounded size of stream data, a full materialization is not possible in some cases.

5. Data stream queries need to run continuously and return new results while new stream elements arrive. The ordering of stream elements may be implicit, which can be in the form of arrival time at the system, or explicit when the stream elements provide an application timestamp with their time of creation. In addition to raw stream data, some use cases need to enrich the data streams with stored data.

A data stream can be generally characterized into one of the following categories:

1. According to Time intervals between packages:

   - Strongly periodic data stream:
     - When the time intervals stay the same length between two packets, then the stream has a strongly periodic characteristic (see figure 3.1). In the optimal case, the jitter has the value zero.
     - Example: Pulse code modulation coded speech in classic telephone switching.

   - Weakly periodic data stream:
     - When the time interval between two packages is not constant but periodical, then the stream has a weakly periodic characteristic (see figure 3.2).
     - Example: Segmented transmission.

Figure 3.2: Illustration of data flow with segments with periodic nature represented by t2

Figure 3.3: Illustration of data flow with with no pattern of the time interval between packages

- A-periodic data stream
  - When the sequence of time intervals is neither strongly nor weakly periodic, and the time period or time gap varies between packets to packets during the transmission, then such data stream is called an a-periodic data stream, like illustrated in figure 3.3.
  - Example: Instant messaging systems.

2. According to variation of consecutive packet amounts

- Strongly regular data stream:
  - When the amount of data is constant during the lifetime of a data stream. This feature is especially found in uncompressed digital or sensor data transmission. Figure 3.4 illustrates this characteristic.
  - Example: Video stream of cameras in uncompressed form, audio stream or ECG sensors.

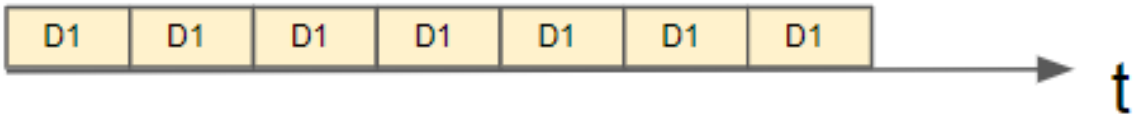

Figure 3.4: Illustration of a constant data size

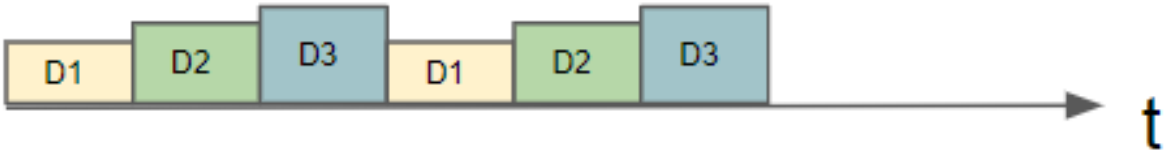

Figure 3.5: Illustration of a weakly regular data size stream

- Weakly regular data stream:
  - When the size of the data stream items varies periodically then it is called weakly regular data stream, like shown in figure 3.5.
  - Example: Compressed video stream.

3. According to connection or continuity between consecutive packets

- Continuous data stream
  - If the packets are transmitted without intermediate gaps.
  - Example: Audio data.
- Unconnected data stream
  - A data stream with gaps between information items is called an unconnected data stream.
  - Example: Compressed video stream.
- Irregular data stream
  - If the amount of data is not constant or changes, then the data stream is called irregular. Transmission and processing of this type of stream are more complicated since the stream has a variable (bit) rate after applying compression methods. Figure 3.6 illustrates this characteristic.
  - Example: Sentiment analysis on trending tweets.

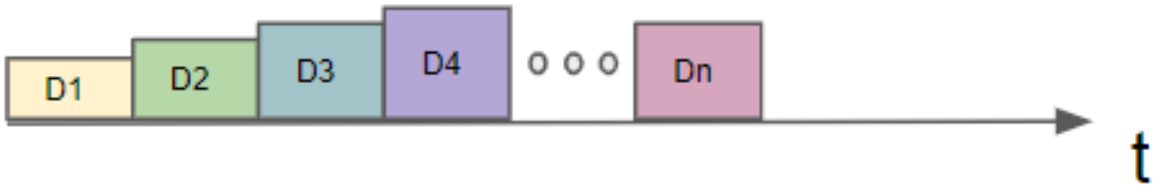

Figure 3.6: Illustration of a irregular data size stream

3.2 IoT Stream processing

Stream processing in the field of connection-oriented communications means to transmit and interpret raw data to convey data packets for the higher-level network abstraction. It is a one-pass data-processing paradigm that always keeps the data in motion to achieve low processing-latency. Stream processing supports message aggregation and delivery, as well as the capability of performing real-time asynchronous computation. The one-at-a-time processing model applies predefined queries or established rules to data streams to get immediate results upon their arrival, which leads to the requirement for simple and independent computations for analytics and pattern recognition.

There are two major classes in the field of stream processing; DSMS and CEP. Data streams within the context of DSMS are joined, filtered, and transformed according to specific logics with the goal of continuous and long-standing queries. State-of-art DSMSs adopt mostly the imperative implementation of long-time queries, where a segment of code is performed upon the arrival of data element for the whole-analysis logic [BD16]. A typical use-case of DSMS includes face recognition from a continuous video stream, calculation of user preference according to the click history or traffic analysis for service providers.

The other use case is called CEP, which is essentially tracking and processing streams of raw events to derive composite events and identify meaningful insights from them. Complex event processing will be further elaborated in section 3.2.5 and showcased in the case study under 5.4.4.

By now, the focus will be held on the DSMS. In the next section, the fundamentals of processing will be described by the lowest level of semantic primitives, followed by examples of more complex processing scenarios.

3.2.1 Semantic primitives for stream processing

In this section, the universal primitive functions of stream processing are explained in detail. The represented operators are explained with their requirements in a streaming scenario. The operators can be categorized into two categories of non-blocking and blocking operators. **Non-blocking** operators can be applied on a constant stream without further requirements while **blocking** operators present results only on a finite sequence.

Non-blocking

Filter: Let P_T be the set of filter predicates over tuples of type T. The filter

$$\sigma : S_t \times P_T -> S_T \tag{3.1}$$

returns all stream elements of a logical stream whose tuple satisfies a given filter predicate. A filter predicate $p \in P_T$ is a function

$$p : \Omega T -> \{true, false\} \tag{3.2}$$

. The argument predicate is expressed as a subscript. The definition of σ_p indicates that the input and output stream have the same type.

Map: Let F_{map} be the set of all mapping functions that map tuples of type $T1$ to tuples of type $T2$. The map operator

$$\mu : S_{T1} \times F_{map} -> S_{T2} \tag{3.3}$$

applies a given mapping function $f \in F_{map}$ to the tuple component of every stream element. The argument function is expressed as a subscript. The mapping function can be a higher-order function. It is therefore sufficient to have only a single mapping function. This definition of the map operator is more important than its relational counterpart as it allows to create tuples of an entirely different type as output. The

projection operator of the extended relational algebra can only project to attributes and add new attributes by evaluating arithmetic expressions over the existing attributes.

Blocking

Union: The union

$$U : S_T \times S_T -> S_T \tag{3.4}$$

merges two logical streams of compatible types. The multiplicity of a tuple e at time instant t in the output stream results from the sum of the corresponding multiplicities in both input streams.

Cartesian Product: The Cartesian product

$$X : S_{T1} \times S_{T2} -> S_{T3} \tag{3.5}$$

of two logical streams combines elements of both input streams whose tuples are valid at the same time instant. Let $T3$ denote the output type. The auxiliary function :

$$\Omega T1 \times \Omega T2 -> \Omega T3 \tag{3.6}$$

creates an output tuple by concatenating the contributing tuples. The product of their multiplicities determines the multiplicity of the output tuple.

Duplicate Elimination: The duplicate elimination

$$\delta : S_T -> S_T \tag{3.7}$$

eliminates duplicate tuples for every time instant. The multiplicities of all tuples are hence set to 1.

Difference: The difference :

$$S_T \times S_T -> S_T \tag{3.8}$$

subtracts elements of the second argument stream from the first argument stream. Value-equivalent elements valid at the same time instant are subtracted in terms of multiplicities. The types of both streams need to be compatible.

Grouping: Let F_{group} be the set of all grouping functions over type T . Let $k \in N, k > 0$, be the number of possible groups for a given input stream. A grouping function

$$f_{group} \in F_{group}, f_{group} : \Omega T - > 1, ..., k \tag{3.9}$$

determines a group identifier for every tuple. The grouping partitions the input stream into k disjoint sub-streams according to the given grouping function (expressed as subscript).

Scalar Aggregation: Let F_{agg} be the set of all aggregate functions over type $T1$. Aggregate function

$$f_{agg} \in F_{agg} with f_{agg} : \mathcal{P}(\Omega \times N) - > \Omega T2 \tag{3.10}$$

computes an aggregate of type $T2$ from a set of elements of the form (tuple, multiplicity). The aggregate function is specified as subscript P denotes the power set. The aggregation

$$\alpha : SlT1 \times F_{agg} - > SlT2 \tag{3.11}$$

evaluates the given aggregate function for every time instant on the non-temporal multi-set of all tuples from the input stream being valid at this instant. The aggregation implicitly eliminates duplicates for every time instant as it computes an aggregate value for all tuples valid at the same time instant weighted by the corresponding multiplicities. Note that the aggregate function can be a higher-order function. As a result, it is possible to evaluate multiple aggregate functions over the input stream in parallel. The output type $T2$ describes the aggregates returned by the aggregate function.

Aggregate consists of the aggregate values and grouping information. The latter is essential if a grouping is performed before the aggregation. An aggregate function should retain the portion of the tuples relevant to identify their group. For the relational case, this portion would correspond to the grouping attributes. Recall that the scalar aggregation treats its input stream as a single group.

3.2.2 Primitive functions - Time-Series analysis

This subsection will illustrate primitive processing in time series more in detail. A time series is a set of periodic and chronological elements and characterized by its numerical and continuous nature. With a high load of data and continuous stream, it can be of interest to reduce the size of the data. Representation methods deal with dimensionality reduction, data transformation, and keeping characteristic features rather than storing whole time series. After retrieving representative features, subsequent stream analysis can be differentiated between three main categories:

1. Indexing: Given a database of stored time series, a query time series, and a similarity measure, indexing can help to efficiently extract the set of the most similar time series from a database.

2. Clustering: Clustering is a technique where data is positioned into homogeneous groups where no explicit definitions nor meta-information on the groups are known. Clustering can help to find groupings of time series data. With a similarity measure present, clusters are constructed by grouping time series that have maximum similarity with other time series within the same group and minimum similarity with time series from the other groups. An example could be the detection of anomalous behavior in the data stream, where no previous knowledge exists of how such an anomalous behavior could look.

3. Classification: With a set of predefined classes present, a database of classified time series, and with the help of a specific similarity measure, an unlabeled time series, when going through the classification procedure will be assigned a known class at the end. An example represents the classification of activities of a sensor streams attached to an athlete.

Besides those three main categories, several more terms are used in the research community, which can be in very general speaking categorized into the aforemen-

tioned categories. The following methods are some of them: Subsequence matching, motif discovery, identifying pattern, trend analysis, summarization, and forecasting.

3.2.2.1 Representation Methods for Dimensionality Reduction

Dimensionality reduction is one of the main goals of the data stream and time-series representation methods. By reducing the dimension, the number of data points that make up a time series will decrease. The challenge lays in preserving the semantic, character, and shape of the original time series. A basic example approach is to sample over a time series randomly, periodically, or follow different criteria to reduce the dimension. Depending on the use case, the global shape and general semantic can be lost, if the sampling window is not specified precisely. Instead of taking sampled data points, other metrics can be of use like Piecewise Aggregate Approximation (PAA) to build averages with adaptive varying-length segments, the Segmented Sum of Variation (SSV) or min-max extraction of a segment.

One of the most used representation methods is the Piecewise Linear Representation (PLR), which averages time series into segments, which leads to an approximation of the original shape. Linear regression can also be used to break a time series window down into segments with its best-fitting line.

The Perceptually Important Points (PIPs) algorithm identifies points of interests. The idea is to preserve the most important and descriptive points within a time series and discards all other points in between. At the initialization of a window, the first data points are considered as PIPs. The following PIPs are identified by the max distance to the other preserved PIPs. Another method is to transform time series data into symbolic strings that represent a pattern of the time series. This type of method can be used in combination with sampling methods like PAA, where an algorithm like Symbolic Aggregate Approximation (SAX) can be used to represent the fixed window of the time series by a predefined alphabet. The aforementioned methods keep the transformation in the time domains. Other methods convert time series into different representations in various domains. A famous example of those methods is the *Discrete Fourier Transformation* (DFT) or Discrete Wavelet Transformation (DWT) and Haar Transform.

Shapelets is a recently introduced concept, where shapelets represents a maximally discriminative sub-sequence of time series data. Shapelets identify short dis-

criminative series segments. Modern methods discover shapelets by analysing subsequences from all possible time-series segments [BC19] [Bag13] and follow by sorting the top performing segments according their target prediction quality. Distances between series and shapelets represent shapelet-transformed classification features for a series of segregation metrics, such as information gain [BC19] [BRK+05], FStat [BBD+02] or Kruskal-Wallis.

3.2.2.2 Indexing

The idea of indexing is to put time series representations into indexing structures to reduce the overhead of the processing. Given a query time series, a set of the most similar time-series is retrieved and returned. The queries could be divided into two types: the e-range and k-nearest neighbors. E-range retrieves all time-series where the distances between them and the query time series are less or equal to e. k-nearest neighbors return to the query time series depending on a specific similarity measure. With that, the k-nearest neighbors' technique can deal with scaling, gaps, and faster rejection of irrelevant candidates[XGP+15][LCW07].

3.2.2.3 Sequence matching

The similarity is only an approximation over a continuous time-series data streams. Two ways of measuring can be considered, namely whole sequence matching or subsequence matching. A popular method in the category of whole sequence matching is the dynamic time warping technique, which extracts patterns from time series for further matching analytics. In Subsequence Matching, two time series of different lengths are matched. The main task is to find subsequences that have the same length and are similar. The sliding window mechanism (explained in section 3.2.4) is ideal support for this specific task [KP00]. Subsequence matching is computationally expensive, and it suffers from bottlenecks regarding the processing time and the disk access time, because of redundant access to store and process resources for the post-processing step that enhances the results [LPK06].

3.2.2.4 Clustering

The distribution of time series over clusters should be carried out in a way that maximizes inter-cluster variance and minimizes intra-cluster variance. Such grouping

mechanisms could help to understand and analyze what knowledge is conveyed in data. Distance-based clustering is extensively used for motif discovery[SA15], anomaly detection [LJGC+17][JNIH16] and finding discords[YKR08]. One of the main challenges when it comes to clustering is to specify a correct number of clusters to capture the (dis)similarity of time series. Time-series clustering could be further divided into two parts, whole series and subsequence.

Whole series clustering: Whole time-series are used to form a cluster. Partioning, hierarchical, model-based clustering are relevant for time series data. Clustering has proven to be very efficient for data streams [CMZ07][CP08][MBSRJ18].

Subsequence clustering: Clusters are constructed from subsequences instead of the complete time series. Time series may vary in their structure over time, where subsequences belong in different clusters. An example could be to use DFT to analyze periodicity of time series and therefore to slice them into non-overlapping chunks. These chunks are tested for similarity and grouped into clusters. Here the questions lay in whether to use overlapping or non-overlapping information to catch important structures and not produce meaningless results. Solutions to these inconsistencies start when not all subsequences were forced into the clustering procedure, but some of them were ignored all together[HYX+16][KR04].

3.2.2.5 Classification

The major difference between classification and clustering is that classification is known in advance. The task is to learn distinctive features that characterize each class. Afterward, the unlabeled data can be assigned to the right class. An important characteristic of the features should be of being non-overlapping distinguishing and discriminative so that the classification can be done more precisely.

Whole series classification Nearest neighbor classifiers are often used to, e.g., classify the unlabeled query with the same class as the most similar labeled time series. This technique can be beneficial if there may be discriminatory features that characterize the whole time series. In its simplest form, Euclidean distance could be used, either on raw time series or transformed representations, like *Piecewise Linear Representation* (PLR), *Symbolic Aggregate Approximation* (SAX) and others. However, this assumes that time series are perfectly aligned, and this is often not the case in real-life situations. Therefore alternative elastic distance measures are usually employed like *Dynamic Time*

Warping (DTW). DTW suffers from some drawbacks when it comes to outliers and data imperfection.To boost classification accuracy, different approaches have enhanced this raw method. In [JKJO11], the authors added a multiplicative weight penalty to reduce the warping.

Interval based classification In the case of noise and redundant shapes, whole series classifiers may get confused and deliver inaccurate results. This is where extracting features from intervals rather than the whole series could be desirable. The challenge to conduct such a technique is finding the best interval. Specifically, there is an undefined number of possible intervals, so how to select the best one and what to extract from each interval once selected? [RAM05] proposed an approach where interval lengths were equal to powers of two, and binary features were extracted from each interval. Afterward, a SVM was trained on the extracted features. A famous interval-based classification approach is the Time Series Forest (TSF)[DRTM13]. It is a random forest approach. After specifying the number of desired decision trees, each tree is trained by dividing the time series into random intervals where n denotes the length of the time series. Effectively the training is done on three features extracted from each interval; the mean, the standard deviation, and the slop. The final classification result is determined by a majority voting.

Dictionary based classification Dictionary-based classification methods are suitable approaches, if motifs or frequent patterns are what characterizes a given class. This technique counts the frequency of patterns. The main idea is to slide a window of a given length and then compute the distribution of words over the different training instances. Following this mechanism, the correlation between the frequency of specific patterns and the occurrence of particular classes could be established. The Bag of Patterns (BOP) approach proposed in [LKL12] computes a histogram of words for time series that are transformed with SAX. Then the same method is followed to build a histogram for unlabeled time series and finally the new series is given the class of the most similar histogram.

Shapelet based classificaiton In scenarios where discriminative patterns could characterize the class of the time series, shapelets are the most suitable technique. They are shapes(subsequences) that occur within the time series, independently from the phase (the placement at the beginning, middle, end). E.g., ECG abnormality detection could be seen in fatal heartbeats in ECG data. Shapelets were devised in 2009 [YJK09], and in their work, authors discover shapelets through testing all possible can-

didates between two given lengths. This approach is called the brute force algorithm. It archives very accurate results but with a high computation cost. The time complexity is $O(n^2m^4)$ where n is the number of time series and m is the length. A more recent approach is the Fast Shapelets (FS) algorithm. This algorithm drastically improved the time complexity to find shapelets. In this context, the computing complexity results in $O(nm^2)$.

Other shapelet based approaches [HLB+14][CMC] formulate what is called *Shapelet Transform* (ST). These approaches are more concerned with the discovery of discriminative shapelets rather than building a classifier to use them. Following the ST, the original time series is transformed into a vector of distances where each element represents the distance between a given time series and a specific shapelet. ST balances the trade-off between the number and quality of shapelets is done by counting on the information gain metric. In the end, the top k shapelets for each class are returned. Another interesting approach that learns shapelets is called *Learned Shapelets* (LS). This proposition learns shapelets that characterize the available classes; however, the learning procedure is different from other algorithms like *Fast Shapelets* (FS) and ST. Traditionally, shapelets are defined as subsequences from the original time series, yet in LS, they are not limited to that. LS uses k-means clustering of candidates in the training data set to initialize k-shapelets. Then these shapelets are adjusted based on a regression model for each class.

Early Classificaiton Early classification usually deals with online stream classification where the interest is to classify as early as possible the currently streamed time series, without waiting for the complete time series to be read. Given the smallest possible sub-sequence of a time series, the main goal is to predict its class accurately. As mentioned, shapelets are shapes that characterize a specific class, and the class could be immediately predicted at any given point in the stream. Furthermore, if shapelets lengths are taken into consideration when attributing utility scores, earliness could be significantly boosted. Approaches such as [MTZ16a] [MTZ16b] prove that shapelets perfectly fit for early classification and rule discovery.

3.2.3 Anomaly detection

Anomaly detection can be found in the fields of:

- Quality control of faulty sensor readings or data corruption during transmission.

- Monitor machine sensors to detect and predict possible failures, before they occur.
- Understand resources usage and power distributions in a connected building.
- Identify anomalous patterns in direction or speed from a vehicle.
- Identify network changes and server degradation.

Algorithms for anomaly detection in a sensor's time-series data can be put in the main categories of statistical, probabilistic, proximity-based, clustering-based, and prediction-based methods:

Statistical methods use measurements to approximate a model. Whenever a new measurement is registered, it is compared to the model and, if it results to be statistically incompatible with it, then it is marked as an anomaly [GMM18a]. Statistical methods can be applied to single elements or window segments. An approximation over a window improves the approximation.

Probabilistic methods describe probabilistic models in parametric or non-parametric nature, depending on the distribution. The classification of anomalies is performed by measuring the probability of an analyzed element or segment. If the probability of distribution falls below a threshold, an anomalous event is detected.

Proximity-based methods rely on distances between data measurements to distinguish between anomalous and correct readings. A popular proximity-based algorithm is the *Local Outlier Factor* (LOF), which assigns an outlier score to each element based k nearest neighbor density measurements [GMM18b]. Readings with high outlier scores are labeled as anomalies.

Clustering-based methods create a measurement between different elements and cluster them based on their similarity. New measurements are assigned to similar clusters or labeled as anomalous in case of a to high distance to existing clusters. Cluster methods and proximity-based methods are usually not the best choices for high dimensional data [GMM18a]. The typical approach is to find the right representation method of differentiating between normal and anomalous data [elaborated in 3.2.2.1].

Prediction-based methods use past measurements to train a model that can predict the value of the next measurement of time-series data. If the actual data has no similarity with the predicted data, then it will be labeled as an anomalous element.

The domain of prediction is increasing steadily with the rise of machine learning. Examples are *Support Vector Machine* (SVM), Deep Neural Networks (DNN), Long Short-Term Memory (LSTM) or HTM (which is applied in the case study). Which algorithm to use strongly depend on the characteristics of the data. The following questions can be taken as considerations for selecting a predictive model:

1. What kind of data is available for training the predictive models? (online vs. offline training)
2. Which format does the data have - are representative features for prediction given?
3. The properties may change over time, which is called concept drift. E.g., will offline learning algorithms be able to learn on drifts?

An example of the HTM online learning algorithm is given in the case study by analyzing anomalous behavior in an ECG signal stream in chapter 5.4.1.

3.2.4 Window methods - bounded stream

The continuous data stream model introduces new challenges for the implementation of queries. Algorithms only have sequential access and need to store some state information from stream elements that have been seen previously, e. g., the join, and aggregation, which must be computable within a limited amount of space on an unbounded stream. This requires approximation techniques that trade output accuracy for memory usage and opens up the question of which reliable guarantees can be given for the output. Like previously mentioned, some implementations of relational operators are blocking. Examples are the difference that computes results by subtracting the second input from the first input, or the aggregation with SUM, COUNT, or MAX.

Stateful/blocking operators (cartesian product, join, union, set difference, spatial aggregation, etc.) require the entire input sets to be completed. These blocking operators will produce no results until the data stream ends. To output results continuously and not wait until the data streams end, blocking operators must be transformed into monotonic queries. A selection operator over a single stream is an example of a monotonic query at any point in time T' when a new tuple arrives, it either satisfies selection predicate, or it does not, and all the previously returned results (tuples) remain in $Q(t')$.

Both standard and spatial aggregate operators always return a stream of length one - they are non-monotonic and thus blocking.

A dominant technique to overcome transforming the blocking queries into their non-blocking counterpart is to restrict the operator range to a finite window over input streams. Windows limits and focuses on the scope of an operator or a query to a manageable portion of the data stream. A window is a stream-to-relation operator that specifies a snapshot of a finite portion of a stream at any time point as a temporary relation. In other words, a window transforms blocking operators and queries to compute in a non-blocking manner. At the same time, the most recent data is emphasized, which is more relevant than the older data in the majority of data stream applications. The following types are being extensively used in conventional DSMS: (Logical) time-based windows and (physical) tuple based windows. By default, a time-based window is refreshed at every time tick, and tuple based window is refreshed when a new tuple arrives. The tuples enter and expire from the window in a first-in-first-expire pattern. Whenever a tuple becomes old enough, it is expired(deleted) from memory. These two window type are not useful in answering an important class of queries over the spatio-temporal data stream, and predicate based window have been proposed in which an arbitrary logical predicate specifies the window content.
In the following, the description of different window methods:

Time-based sliding window

Time-based sliding windows are characterized by two parameters: The window size and also the window slide. The size is defined in terms of the time length and says that the window content at time t contains only the elements with a timestamps bigger than t-size. The slide parameter, fixed as a time length, indicates how often the window is going to be computed, or slided over time.

Time-based tumbling window

In this case, the size of the sliding window is equal to the window length. For instance: Give the room where person X has been within the last ten minutes; change results every ten minutes.

Triple-based windows

Triple-based windows were characterized to emulate tuple count windows in *Continuous Query Language* (CQL)-like data stream systems.

Count-based window based on Basic Graph Pattern

Basic Graph Pattern (BGP) count-based window is introduced to deal with the above

limitation of a triple-based window on *Resource Description Framework* (RDF) streams characterized as sequences of graphs [W3C16]. Rather than counting single triples, this count-based window can count the groups of triples (subgraphs) that match a specific basic graph pattern.

Partitioned Windows

Deal with one input stream and a number of other output streams (i.e., partitions), over that the query is evaluated. The partitioned windows are based on knowing the underlying schema, and deciding of how to do the partition in an effective way for the query.

Predicate-based window

Predicate-based windows, qualified objects are a part of the window once they fulfill a particular query predicate. Objects expire if they no longer satisfy a particular predicate and are a generalization of tuple-count and time-based windows [W3C16].

3.2.5 Primitive- vs Complex-event processing

In contrast to the primary goal of DSMS, which performs stream analytics at a geographically concentrated place, the major concern of CEP is to infer the global meaning of raw events to stream as fast as possible [HPX11]. An event object carries general metadata (event ID, timestamp) and event-specific information like, sensor-id, measurements. Atomic events might have no special meaning but can get a higher meaning in correlation with other events. CEP analyses continuous streams of events in order to identify complex sequences of events (event patterns). A pattern match represents a relevant state of the environment and causes the trigger to creating a new complex event or triggering an action.

Event Processing Language (EPL) is a fundamental concept of CEP, which contains event processing rules defining event patterns, actions, and event processing engines to analyze events and match rules continuously [BBDR13].

CEP engines usually have the following basic characteristics:

- Continuous in-memory processing: Designed to handle a sequential input stream of events and in-memory processing enables real-time operations.
- Correlating Data: Enables the combination of different event types from heterogeneous sources. Event processing rules transform primitive events

into complex events that represent a significant meaning for the application domain.

– Temporal Operators: Within event stream processing, timer functionalities as well as sliding time windows can be used to define event patterns representing temporal relationships.

CEP provides a rich set of concepts and operators for processing events, which include the CQL like, queries, rules, primitive functions (aggregation, filtering, transformation, etc.) and production of derived events. The events are manipulated by CEP rules, which are *Event-Condition-Actions* (ECA). ECA is a combination of continuous query primitives with context operators. Context operators can contain the characteristics of temporal, logical, and quantifiers, where a positive correlation generates a complex event that summarizes the correlated input. [EHPdAS16].
The following points represent phase examples of a CEP processing flow:

- **Signaling**: Detection of an event.
- **Triggering**: Check an event against a defined set of rules.
- **Evaluation**: Evaluate the condition of each checked rule.
- **Scheduling**: Execution order of selected rules.
- **Execution**: Execute all actions for the relevant selected rules.

Rule Model can be classified into two main classes - transforming rules and detecting rules. **Transforming rules** can be considered as a graph connection of primitive operators. The operators take multiple input flows and produce new elements that can be forwarded to other operators or consumers. Transforming rules can be usually found with homogeneous information flows where the structure of the input and output can be anticipated [CM12a]. **Detecting rules** present an distinction between a condition and an action. The condition part represents a logical predicate that captures patterns in a sequence of elements. The action part defines how the information gets processed and aggregated.
The language types of CEP can be divided into the classes of transforming and detection based languages [CM12b]:

- **Transforming languages** defines transforming rules on input streams by filtering, joining, and aggregating received information to produce new output flows [CM12a]. Transforming rules can be divided into two sub-classes:

  - **Declarative languages** express processing rules by specifying the expected results, instead of the execution flow.
  - **Imperative languages** define rules that specify a plan of primitive operators. Each operator contains a transformation over its input.

- **Detecting or pattern-based languages** specify the trigger conditions and the actions to be taken when specific conditions hold. Conditions are represented by patterns over the input stream and constructed with logical operators, timing and content and constraints. Actions define how the detected events have to be combined to produce new events. This is a common language in CEP systems, that has the goal to detect relevant information items before the evaluation process is started.

Modern languages consolidate operators of different languages. An example for declarative languages is CQL. [ABW06]

3.2.6 Primitive functions for CEP

CQL defines three classes of operators [CM12c]:

- **relation-to-relation operators** define classical queries over database tables and are similar to SQL queries.
- **stream-to-relation operators** create tables out of a bounded segment of stream data. Relation-to-stream operators create a data flow out of fixed tables.
- **Stream-to-stream operators** are expressed using the other operators.

Stream-to-relation operators are based on sliding window operator concepts. The relation is obtained from a stream by extracting the set of tuples included in the current window.

CQL contains 3 **relation-to-stream operators**:

- Istream(): whenever a tuple t is inserted into the input relation at time $\tau \rightarrow Outputs(\tau$, t)
- Dstream(): whenever a tuple t is deleted from the input relation at time $\tau \rightarrow Outputs(\tau$, t)
- Rstream(): whenever a tuple t is updated in the input relation at time $\tau \rightarrow Outputs(\tau$, t)

Example:

```
SELECT  Istream(*)
FROM    S  [Rows 100]
WHERE   S.A \rightarrow 2
```

Execution explanation:

1. Source of the query is the referenced stream S.
2. Stream-to-relation operator [Rows 100] converts the input stream of 100 entries into a relation.
3. Relation-to-relation filter S.A =¿2 selects the attributes with values higher than 2, which results in another relation.
4. Istream relation-to-stream operator transforms the output back into a stream.

Pattern-based languages use selection operators to find items or events of a complex pattern. In the publish-subscribe system, they are the main operators of choosing items to be forwarded to consumers [MC11].
The following logical primitive operators are used in a CEP system:

- A **conjunction** of elements $E_1, E_2, ...E_n$ is satisfied when all the elements $E_1, E_2, ...E_n$ have been detected.
- A **disjunction** of elements $E_1, E_2, ...E_n$ is satisfied when at least one of the elements $E_1, E_2, ...E_n$ has been detected.
- A **repetition** of an element E of degree m, n is satisfied when E is detected at least m times and not more than n times.

- A **negation** of an element I is satisfied when E is not detected.

Logic operators are present in pattern-based languages, where they combine different. Declarative and imperative languages do not provide logic operators explicitly. They allow con(disc)junctions and negations using rules that transform input stream.

Sequences

Sequences are used to define the arrival of a set of elements where the order of arrival is considered. A sequence defines an ordered set of elements $E_1, E_2, ...E_n$ which is satisfied when all the elements $E_1, E_2, ...E_n$ have been detected in the same order.

Parameterization

Parameterizations define the capability to filter some data streams based on elements that are part of other streams. Use Case: A fire alarm of a building is being notified when the temperature of a room is over a defined threshold of 41°C, but only if smoke has been detected. Declarative and imperative language solution: Joining two information flows with temperature and smoke within same room. Pattern-based language solution: Do not provide a join operator but can use the conjunction operator to capture the single events of high temperature and high smoke in the same room via parametric filtering on the streams. **Aggregates**

Many CEP applications need to aggregate the content of multiple, incoming information items to produce new information.

Most languages have predefined aggregates, which include minimum, maximum, and average. In declarative and imperative languages, aggregates are usually combined with the use of windows. Pattern-based languages include detection aggregates to capture patterns over stored values.

3.3 Semantic IoT Stream enrichment and reasoning

This chapter details the process of annotation of sensory data with semantic descriptions specifying spatial, temporal, and thematic attributes. Section 3.3.1 explains the semantic transformation and annotation process in which the delivered IoT data is being transformed from a middleware message into a bag of metadata. The annotated data provides a basis for interoperation among different components using machine-interpretable language (i.e., RDF and Turtle). Section 3.3.2 presents an information model for semantic modeling of the streaming data, as an extension of the SSN ontol-

ogy which allows representing not only sensor observations but also aggregated values and temporal entities such as segments and data points involved in a data stream. Section 3.3.3 shows a comparison of different CEP processing methods for semantic data.

3.3.1 Semantic stream processing concepts

Stream Transformation

This component aims to transform a message into a bag of metadata to be used in the semantic annotation process. In the context of the Semantic Web, syntactically defined messages that are provided by IoT applications need to be conceptually re-engineered in order to extract the semantics of the intended actions and the underlying IoT domain concepts. This requires not only structural transformation but also semantic transformation. The ultimate result of the semantic transformation is a set of semantic models, which are used for annotation.

Semantic Annotation of IoT Ressources

In IoT applications can be a need to use ontologies to represent the domain knowledge in order to deal with various forms of heterogeneity. This heterogeneity can be in the form of data modality and data representation. Utilisation of semantic technologies for IoT enables interoperability between IoT resources, representation models, data providers, as well as consumers. Semantic Web technologies provide solutions for this challenge with standardized knowledge representation, information sharing, and interlinking of heterogeneous resources.

Sensors and their data can be linked to geographic data (e.g., correlated natural phenomena), user-generated data, and general data (e.g., public transportation, health data) through a knowledge-based approach.

Semantic Interoperability

With semantic interoperability, data can be accessed by different stakeholders in a generalized representation. IoT entities need to communicate data between each others, with other users or need to be related in more complex representations. Unambiguous data descriptions are a key enabler of automated information communications, interactions, and complex reasoning. Semantic annotation of the data with domain knowledge can provide machine-interpretable descriptions of the data source, the relation to other entities, and the meaning in a more complex context [BWHT12].

3.3.2 Linked Data Components

Resource/service search and discovery

Search and discovery mechanisms are an essential function to locate resources and services to relate them to entities. To support the search and discovery, the principles of linked sensor data is used to enable publishing and usage of sensor data.

IoT data abstraction and access

IoT Data abstraction deals with how the generated data is represented and managed. Ontologies such as the W3C's SSN ontology [W3C14] have been developed to represent sensor data. Ontologies are constructs to formally represent sensor resources and their generated data on different abstraction levels. Data access can be implemented at low-levels (e.g., device or edge level). Heterogenous device and network environments make data access difficult, where sensing as a service is a scalable way to access data.

The semantic Web technologies include standards (e.g., RDF or OWL) and tools for creating and querying semantic data. The semantic annotation supports more effective mechanisms to integrate the IoT data, however, an autonomous and easy integration still needs effective reasoning and processing mechanisms [BWHT12].

The booming state of IoT, as well as the wide range of IoT platforms, standards, and custom implementations, create challenges to define the right semantic schema. RDF is recommendation by *World Wide Web Consortium* (W3C) for structuring and representing information.[W3C13]

The benefit of RDF schemas can be summarized in three aspects:

1. The RDF model consists of triples and due to that be efficiently implemented and stored. Other models with variable-length fields might require a more complex implementation.

2. The basic RDF model can be processed in the absence of more detailed information (RDF schema) on the semantics. It allows basic inferences, since it can be logically seen as a fact knowdledge base.

3. The RDF model is modular, where directed graphs (union of knowledge) are mapped into the corresponding RDF structures, which means that:
    - information processing can be parallelized. in the presence of partial information inside a volatile environment.
    - the output is a consistent RDF model.

A RDF stream is a sequence of RDF graphs that including the describing metadata. The metadata acts as a flexible mechanism to additionally add time-relation information. An example of time metadata are:

- Production time: Time the data element was initially created.
- Receiving time: Time the data element was made available with the RSP engine.
- Start time and finish time: Time when the data element was validated and ended.

The selection of the right timestamp depends on the use case. In many use cases, the system arrival timestamp is enough, while in other cases a validity interval can be of high importance.

This RDF model standard relies on a traditional persisted-data paradigm, in which the main focus is on maintaining a finite set of data parts in a knowledge base. In conclusion, this paradigm does not match the case of general data streams, in which stream elements flow continuously over time, forming infinite sequences of data.

Semantic reasoning and interpretation

Inference algorithms are usually implemented within reasoners like FACT++3 and Jena4 [BWHT12]. As an example, SPARQL is used to construct queries to explore semantic descriptions. SPARQL builds upon the previously mentioned RDF standard [W3C14]. SPARQL is designed to execute queries over RDF triples but does not have the functionalities to query RDF streams. Continuous RDF query languages have introduced to face this challenge. Continuous SPARQL (C-SPARQL) was an early extension of SPARQL for querying RDF streams and supports continuous queries, which are registered and continuously executed while considering stream windows [BBC$^+$09]. However, C-SPARQL is not designed to handle large volumes of data.

SPARQLStream [CCG10] was inspired by C-SPARQL, but only supports windows operators. The stream-to-ontology mapping is done via S2O and R2RML. However, it does not support querying on both stream and RDF datasets. Event-Processing-SPARQL (EP-SPARQL) uses a black-box approach with a logic engine, where queries

are translated into logic programs [AFRS11]. EP-SPARQL unifies event processing and stream reasoning in a logic programming paradigm, where the disadvantage lays in its decreased performance over complex queries [LCL16]. Continuous Query Execution over Linked Stream (CQELS) [PDTPH11], is an adaptive execution framework for Linked Stream Data and Linked Data. CQELS provides a flexible architecture for implementing efficient continuous query processing engines over Linked Data Stream and Linked Data.

CQELS, C-SPARQL, SPARQLStream do not provide any temporal pattern matching operator and thus cannot be classified under the *Semantic Complex Event Processing* (SCEP) languages [GZPL18]. While frameworks like the SPARQL extension EP-SPARQL exist with sequence constructs that allow temporal ordering over triple streams, it does not support the integration of multiple heterogeneous streams and only with single-stream models [ARFS12]. SPAseq extends SPARQL with new SCEP operators which can be evaluated over RDF graph-based events [GZPL18].

3.3.3 Semantic reasoning methodologies

Data streams must be fused into tangible facts by combining information with background knowledge to infer new knowledge. Several states-of-the-art reasoning methods were described in chapter 2. The following table shows a comparison of the pro and cons of the mentioned methods [SG17]:

Table 3.1: IoT Data enrichment Reasoning methods comparison

Methods	**Pro**	**Con**
Logic/rule-based	simple rules, adapted to simple sensors, easy for beginner (learning and implementation)	not adapted to complicated sensors, heterogeneous rule languages and editors
Continued on next page		

Table 3.1 – continued from previous page

Methods	Pro	Con
Machine learning	More elaborate results, adapted to complicated sensors	need real datasets, complicated for non-experts, complicated for a "sharing and reusing" approach
Linked Stream processing	Real-time data, scalability, linked data	no real reasoning
Re-use domain knowledge (LOD, LOV, LOR)	Sharing and reusing approach	complicated for implementation
Distributed reasoning	saclability, interoperability between systems	complicated for implementation
Recommendation systems	adapted to the user profile	complicated for non-experts, need real datasets, need user profile

Logic rule-based reasoning

Several mechanisms and tools have been developed in order to apply processing logic over streams. *Sensor-based Linked Open Rules* (S-LOR) approaches to share and reuse rules for the interpret IoT data based on an interoperable dataset of rules. Domain experts manually wrote the rules and extracted from *Linked Open Vocabulary* (LOV) for the LOV4IoT dataset [GAB+16] [GBBS16] in the fields such as healthcare, smart homes, and smart cities. *Linked Edit Rules* (LER) is an approach similar to S-LOR to share and reuse the rules associated with the data, while it was not applied in the IoT context. LER focuses on checking the consistency of data and extends the RDF data model by introducing the concept of EditRule [MPGS15].

Linked Open Rules

Linked Open Rules (LOR) is an approach for sharing and reusing ways to interpret the data and to infer new information. LOR can be extended towards using S-LOR, that contains a dataset of interoperable rules to interpret sensor data [GBB14] via a triple store. This approach is inspired by the concept of Linked Rules [KJK11], the idea of

semantic rules interchanging in languages.

Machine learning

The papers of [SHSS08] and [WB09] proposes reasoning over semantic sensor data. [WB09] describes the use of "semantic perception"[HST12] and [BGHS12] to interpret and reason over sensor data. This work developed a semantic-based approach to integrating an abductive logic framework and Parsimonious Covering Theory (PCT) for the integration of semantics in sensor devices. A conclusion in this work was that the construction of background knowledge is difficult. For this reason, the LOV4IoT dataset has been designed to encourage the reuse of the domain knowledge expertise relevant for IoT. Logic-based reasoning is faster and more flexible to use for simple sensors like temperature, while complex sensors like ECG still rely on data mining approaches. [GBC13] and [GBC14] introduced a *Knowledge Acquisition Toolkit* (KAT) to infer abstractions on a high level from sensor data in order to reduce traffic overload in network communication. KAT proposes the use of domain-specific background knowledge, which is not sufficient for the IoT unless another approach like the LOV4IoT dataset is used in combination.

[MPP+10] emphasizes the use of machine learning on sensor data with the use of decision trees and Bayesian networks to analyze sensor measurements. The sensor data was enriched with semantic context information, based on ontologies like Geonames for location, Geo WGS84 for coordinates, W3C *Semantic Sensor Network* (SSN) ontology for sensor description, as well as the W3C time ontology [Mor13][MM12]. The in [MMV+11] proposed SemSense architecture is an approach to collect and publish sensor data as Linked Data.

[DK12] designed an ontology for weather events observed by sensors such as wind speed and visibility, where high-level abstractions are deduced with rule-based reasoning via Semantic Web Rule Language (SWRL). [RGSE14] explains the need for a domain-specific automated reasoning system and imagines such a system for interpreting IoT data based on *Description Logic* (DL) or CEP.

Chapter 4

IoT Architecture

According to Cisco, it is estimated that there will be around 50 billion connected devices by 2020 [Eva11]. Current mobile network architectures already face challenges of managing the size and rate of generated data. In many implementations of cloud-based applications, the data is sent to cloud data centers [RP17]. Moving the data from the IoT source to the cloud might not be efficient or even infeasible in many cases due to bandwidth limitations. Further does time-sensitive, or location-aware applications will have the requirement of ultra-low latency in which a distant cloud processing will not represent an option [ZMK+15]. Furthermore, do privacy concerns over generated user-data to cloud services represent a significant challenge.

To address the issues of high-bandwidth, geographically-aware, ultra-low latency, and privacy-sensitive applications, a modern computing paradigm located closer to the source has been proposed. Fog and edge computing address some issues by enabling computing, storage, networking, and data management close to IoT devices [BMZA12]. Other emerging computing paradigms are mist computing or cloudlets, also address these challenges. A visualization of the different computing paradigms can be seen in figure 4.1, which are mentioned in this chapter.

Edge computing systems distinguish between “disposable data” and “critical” data. IoT-capable devices produce large amounts of data with their sensors. With this, the edge acts as a hyper-responsive mediator layer between cyber-physical systems and the data center. The prioritization of data transfer can take place on the spot, and therefore in real-time with minimal network load. The fog is a system-level edge-computing architectural pattern, with specialized gateways and other Fog nodes. The

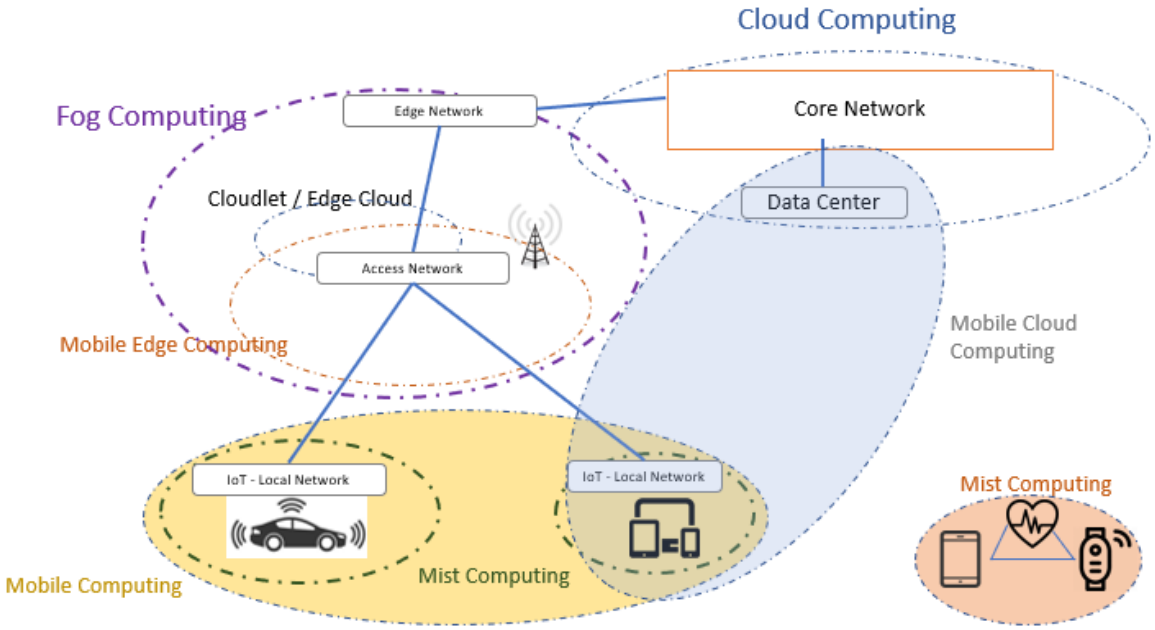

Figure 4.1: Overview of computing paradigms. Reduced version of [YFN+19].

edge computing model processes the data directly on the end devices themselves. Devices as small as Raspberry Pis can handle data processing for a wide range of "smart" IoT endpoints-but their performance is barely scalable, and their availability is barely guaranteed.

4.1 Decentralized architecture paradigms

Fog Computing: Fog Computing implements a decentralized Computing infrastructure based on *Fog Computing Nodes* (FCN) placed between the end devices and the central system. FCNs are heterogeneous and are based on different types of hardware including routers, switches and access points.

Mobile Computing: The development of mobile computing influences the advancement in fog and cloud computing. Mobile computing, is when computing is performed via mobile or portable devices. Mobile computing can be used to create context-aware applications, such as location-based reminders. Mobile computing holds the vision for adaptation in an environment of low processing power and sparse network connectivity. Using mobile computing solely is not suitable for many modern IoT use cases, due to the evolving requirements of connected devices. Fog and cloud computing enables computing outside of local network and expand the scope and scale

of mobile computing. Mobile computing requires only mobile devices, which can be connected through WiFi, Bluetooth, and other cellular protocols. Compared to fog and cloud computing, mobile computing is more resource-constrained. Distributed applications benefit from the distributed mobile computing architecture because devices do not need a centralized location. However, mobile computing comes with drawbacks such as substantial-resource constraints, the balance between autonomy and interdependence, communication latency, and the need for mobile clients to efficiently adapt to changing environments [Sat96]. These drawbacks often make mobile computing unsuitable for current applications that require low-latency or robustness or to deal with large amounts of data.

Mobile Edge Computing: *Mobile Edge Computing* (MEC) can be considered as edge computing implementation that brings computational and storage to the edge of the network to improve context-awareness and reduce latency. The MEC servers are often co-located within the Radio Network Controller or base station. MEC offers real-time information on the network while providing information about connected devices, like location information.

***Mobile Cloud Computing* (MCC):** As cloud computing matured, MCC became a valuable complement to mobile computing. The data storage and data processing get offloaded outside of the mobile device [DTLNW13]. Some MCC applications include healthcare, sensor data processing, and task offloading [RZZS15] [SAGB14]. Mobile applications can be partitioned at runtime to adaptively offload computationally intensive components of the application [SGKB13]. Resource-constrained mobile devices can leverage cloud resource services. MCC shifts computation from mobile devices to the cloud, which can increase the battery life of devices. Yet, this introduces connectivity and latency challenges for delay-sensitive applications.

***Mobile ad-hoc Cloud Computing* (MACC):** The MCC computing paradigm is not always applicable for scenarios in which there is a deficit of infrastructure or a centralized cloud. MACC consists of nodes that form a dynamic network and represent the most decentralized form of networks [HGBV00]. Mobile devices build a dynamic network topology, where the devices continuously join or leave. MACC can include use cases such as disaster management or unmanned vehicle systems.

Cloudlet Computing: A Cloudlet can be dened as a trusted cluster of nodes with resources available to use for nearby mobile devices [BOE17] [SBCD09]. Cloudlet computing has similarities with MCC and MEC. Cloudlet nodes are mini clouds that are typically one hop away from mobile devices and provide resources for mobile device offloading mechanism [HBZZ17]. Operators for cloudlet computing can be cloud service providers who want their services available close to mobile devices to lower latency and energy consumption on mobile devices. MACC and cloudlet computing support mobility, while MACC is resource-constrained on the mobile devices.

Mist Computing: Mist computing had the purpose to capture an endpoint edge of connected devices. This computing paradigm describes dispersed computing on the IoT devices and has been proposed with the idea of self-aware and autonomic systems [PTJ+15]. Mist computing extends with the end device the compute, storage, and networking across the fog. With these characteristics, mist computing is a superset of MACC, since the networking may not be necessarily ad-hoc, and the devices may not be mobile devices. The authors in [MPSRS+17] introduce the idea of using mobile devices as a cloud computing environment for storage, and computing. Some advantages of mist computing are to preserve the privacy of user data by local processing and analysis [SN16].

4.1.1 IoT architecture components

An IoT architecture can be generalized in the now following components.

Sensing layer: It consists of sensors attached to physical devices. These sensors generate data continuously. The generated data from these multiple sensors can be heterogeneous. The sensing layer primary goal is to identify any state in the device sensor and obtain data from it.

Ingestion Layer: This component acts as a message queue for raw data streams that are pushed from the sensing layer. Multiple sources can continuously push data streams (e.g., sensor or social network data). Such a component must be able to deal with high throughput rates and scale according to the number of sources. One of the key responsibilities is to enable the ingestion of all incoming data. This component does not require any knowledge about the data or schema of incoming data streams. How-

ever, for each element, it must know its source and type. To assure fault-tolerance and durability of results in such a distributed environment, techniques such as write-ahead logging or the two-phase commit protocol are used, that have an impact on the availability of data to next components.

Stream Processing: The Stream Processing component is responsible for performing One-Pass algorithms over the stream of elements. The presence of a summary is required as most of these algorithms leverage in-memory stateful data structures. Such data structures can be leveraged to maintain aggregates over a sliding window for a specific time period. Different processing strategies can be adopted, being the most popular tuple at-a-time and micro-batch processing, the former providing low latency while the latter providing high throughput.

Ingestion/middleware Layer - A middleware layer provides between application software and system software.

Network Layer: The network layers role is a communication medium to transfer data, from the sensing layer to other components. IoT devices use different communication technologies (e,g, WiFi, cellular network).

Processing and Actuation Layer: The data processing layer consists of the primary data processing unit. The processing layer takes collected data in the sensing layer and processes or analyses the data to make decisions based on the result.

In the following will be illustrated the most popular IoT stream processing infrastructures:

Traditional cloud-based processing architecture: This archtiecture is based on the IoT device and the upper cloud layer. In this architecture, the IoT devices communicate between the cloud server via a local area network (LAN), mobile network, or wide area network (WAN). The energy consumption needs to be relatively high to enable uninterrupted performance. Due to this, these IoT devices are often stationary and connected to a continuous power supply. Static IoT sensor provides and consumes data occasionally, such as measuring temperature, or controlling access to a door, a small quantity of data is exchanged between the cloud and the IoT device [GKK+19b].

A problem arises with large quantities of data, generated at high speed, that needs to be transferred to the cloud. Traditional cloud-based architecture cannot cope with the increased demand for data transfer. This architecture might not be able to support an energy-efficient solution, in the case of wireless IoT devices [GKK+19b].

Vertical Edge Processing architecture: The previously presented architectural solution cannot cope with the situation when the edge device cannot perform the required services, e.g., in the case of mobile edge devices with limited resources and wireless connectivity. To reduce or balance energy consumption on IoT devices, the edge device will want to distribute the processing task to other devices on the edge. An architectural solution that supports such offloading is the vertical edge computing architecture. An edge server is on the next hope above the edge device and performs the data storage and complex operations. With this, the load on the edge device is reduced and balanced. This edge server can be in the form of a raspberry, mini-cloud, cloudlet server, or mobile computing clusters.

Horizontal Edge Processing architecture: A more promising solution is based on deviceless edge architecture. This architecture does not need the management of edge devices and servers, and offloading is realized horizontally to nearby edge devices.[GKK+19a]

4.2 Decision criteria for IoT architecture selection

To find the right archtitecture for an IoT application, several considerations about the environment limitations should be analysed. A general selection survey could be considered under the following points [DD17]:

1. **Proximity**: Defines the logical- (how many hops) and physical-distance (actual distance to next layer). The lower proximity as a negative influence on the latency.

2. **Access Medium**: Access mechanisms are important to determine the bandwidth available to the end devices, the distance of connectivity and support for different types of devices.

3. **Context awareness**: Further information about device and environment, like, e.g., the device location or network load.

4. **Power consumption**: The power consumption is a major contributing factor for resource-constrained end devices. The consumption with e.g. LTE and radio networks is higher than the energy consumption for WiFi [HQG+12]. The consumption by accessing Mobile Edge nodes is bigger than the consumption of accessing Cloudlets. On the other hand, FC allows access to its nodes through access mediums that consume lower energy like BLE.

5. **Computation time**: defines the required time at the Edge layer for performing the tasks and responding with the desired results. The computation time for MEC and Cloudlet Computing proves to be beneficial due to the virtualized nature along with dynamic resource provisioning. However, since the Fog devices are often legacy devices, the processing and storage capacities are lower, leading to higher computation time.

Chapter 5

Case Study: Scalable IoT data processing and reasoning ecosystem in the field of Health Analytics

In this chapter, the case study is introduced within the context of health analytics and the requirements for a complex event ecosystem in section 5.1. Based on the requirements, the conceptional design of the architecture components follow in section 5.2. Section 5.3 contains the detailed insight in the technical realization of the data processing pipeline. The implementation of the anomaly detection, classification as well as complex event processing follows in section 5.4. The final section 5.5 describes the possible semantic representation schema that can be used for unified representation, as well as integration in already existing knowledge bases.

5.1 Introduction

ECG-stream processing use case: Patients with chronic diseases such as heart conditions or other medical conditions, usually need to frequently pay visits to physicians or are possibly hospitalized for monitoring. This care service can be very impractical for the patient and expensive for hospitals. In some cases, chronically ill patients receive home-based healthcare service from caregiver and doctors. These services are more and more improved with health-monitoring devices which deliver individual analytics results to the care service. The review of this data can help to improved care or alert of acute health conditions. Waiting for a batch analysis of the patient data or a

manual review can leave the patient at risk in critical situations.

Personalized Analytics: Fusing CEP with medical data enables advanced and personalized analytics for each patient in real-time. The reasoning over composite events with the integration in existing health care knowledge bases, offers many use cases to support real-time decision making in emergency situations. Slight differences in a patient's blood pressure or heart rate might not raise any alarm. In a correlated analysis, this can indicate possible heart failure or other critical situations, where a fast prescriptive analysis could recommend specific medications or interventions. A caregiver could be alerted with the specific information and recommendation for medication based on the patient's situation and historical data, eliminating the need for additional extensive manual review. The inference can be retrieved out of a knowledge base to give information about the best suitable treatment method for a specific patient.

A particular focus in this case study will be held on the monitoring and the integration process of the detection of *Cardiovascular disease* (CVD). According to the World Health Organization, a significant number of deaths are caused by CVD [WHO]. Arrhythmia is a type of CVD that relates to any irregular change of the heart rhythms. Although a single arrhythmia heartbeat may not have a severe effect on life, continuous arrhythmia beats can indicate an incoming fatal situation. Prolonged premature ventricular contraction beats can turn into ventricular tachycardia or ventricular fibrillation beats, which can immediately lead to heart failure. Due to that, it is essential to periodically monitor the heart rhythms to prevent and interfere with the CVDs.

Methodology

The result of this case study will be a proposal of practical course of action to process IoT health data in a scalable IoT architecture by evaluating different implementation decisions based on the research fields of data processing, data enrichment and (semantic-) CEP. The methodology of exploratory research will be followed, which aims to explore and evaluate the identified theoretical background by applying a complex IoT use case in the field of e-health.

The project milestones of an IoT use case are generalized in the following steps:

1. Define a processing pipeline for IoT-based communication systems, semantic data acquisition, continuous query processing, complex reasoning, and detailing the implementation of each component.

2. Provide a semantic information model for the representation of IoT data, patient data, and medical domain knowledge in a scenario, by reusing existing semantic models.

3. Define an stream reasoning component as part of the eco-system, based on continuous query processing.

The proposed IoT stream architecture will have generalized requirements expected to be the minimum standard in a healthcare domain scenario. It is important to mention that the defined requirements are not based on a factual requirement analysis with healthcare domain experts. The proof of concept contains generalized requirements for the full integration of semantic enrichment and complex event processing over IoT data streams. The following generalized requirements were defined for the proof of concept:

1. **Requirement 1 - IoT integration in central analytics infrastructure**: Many traditional health monitoring devices have integrated and isolated monitoring functionalities, where an integration outside of the device itself is often not possible. This is shifting with modern IoT devices, that can be integrated into other services. A central analytics environment should be able to collect and correlate IoT data stream events.

2. **Requirement 2 - Real-time alarming system**: A fast reaction time should be made available on detected issues as soon as possible close to the source. With more powerful devices, the architecture design should be flexible enough to offer edge computing to decrease the latency of primitive reasoning over single event streams and forward events to the central complex event processing system if needed.

3. **Requirement 3 - Integrate device analytics in more complex analytics scenarios**: This requirement contains complex event processing over several stream elements to offer an in-depth analysis of a diagnosis that consists of composite events, as well as the integration of CEP inferences on patient data and knowledge base expert systems.

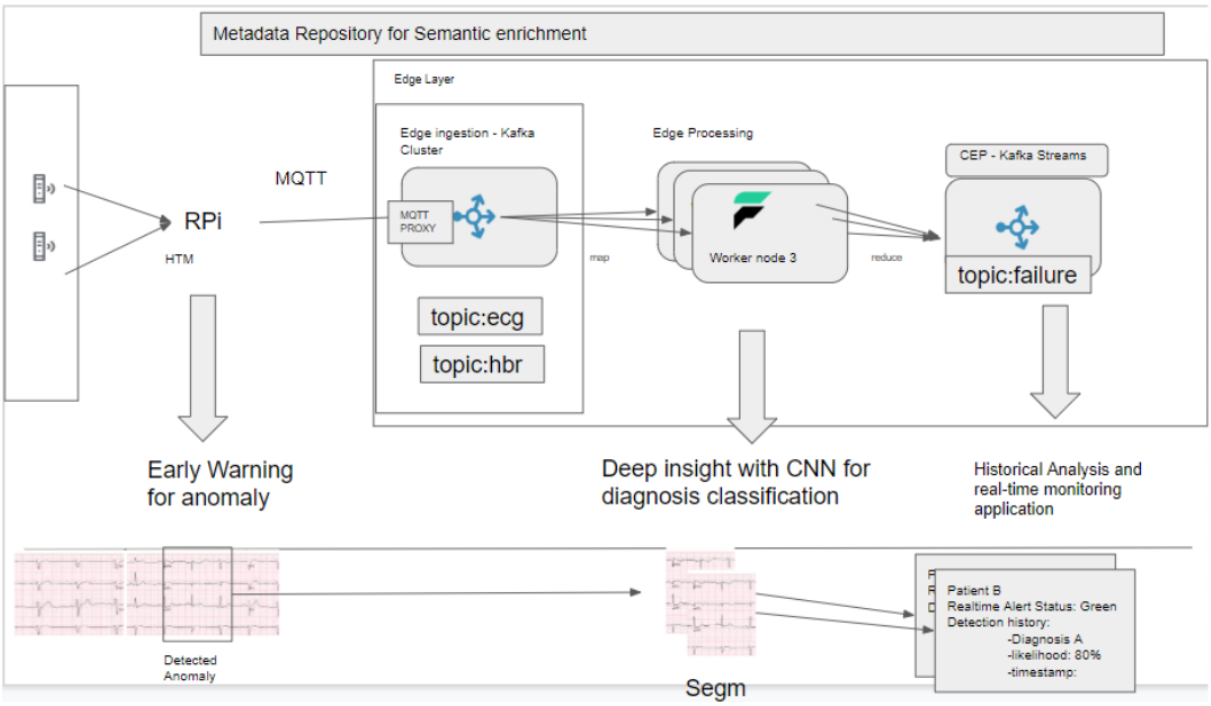

Figure 5.1: Complete processing architecture starting from the IoT data generation, the Kafka ingestion, the following processing via the Faust workers to the event classification

4. **Requirement 4 - Scalable and secure infrastructure**: A scalable and reliable messaging architecture is of utmost importance in the field of healthcare.

5.2 Conceptual Design: Architecture

Conceptual design is the intial design step, in which the general outlines of function and the Eco-system was articulated. The main components are explained in the following subsections, where figure 5.1 shows the overview of the proposed Eco-system.

5.2.1 Ingestion and communication system - Kafka

The first component to be analyzed was the stream ingestion layer.

Kafka is a high throughput, distributed log that can be used as a message queue system. Any number of producers and consumers can be handled by Kafka.

With this principle, Kafka provides persistency for the messages and offers the following characteristics [Kafa]:

- Reliability is the highest requirement
- Some messages should be kept a copy of, even after consumption
- Data loss is not acceptable
- Speed is not the biggest concern
- High data size

The main requirement for a health care system is reliability and data loss prevention, which made Kafka the choice for the implementation of this case study.

Apache Kafka is a streaming platform that allows users to publish data and subscribe to different stream topics. Kafka stores the streams in a fault-tolerant way. A Kafka system runs as a cluster and follows the concept of a topic in which data is published. In this system abstraction, every raw data stream type is published to a topic. This enables to structure streams by different topics, e.g., ECG, Heart Rate, light sensors. Each topic records all the messages for a specified retention period. This capability can be used not only for real-time singles stream analytics but also for health condition analytics in complex event compositions that require using historical patient data.

Different IoT devices act as data producers. Devices publish the raw and early pre-processed data in the specific Kafka cluster, where Kafka consumer can publish back the results of their processing or analytics into the cluster. Kafka manages the scalability of the system as the general load on the architecture may change with an increase in patients. Kafka serves as data retention, storage, and forwarding interface, where several consumers can consume the data.

5.2.1.1 Kafka Architecture

The Kafka architecture is a cluster consisting of several components. The figure 5.2 illustrates the different Kafka components, which are described in this section.

Kafka Broker: A Kafka cluster consists of multiple brokers that enable load balancing. Kafka brokers are stateless, where the ZooKeeper maintains the cluster state. A Kafka cluster always has a leading broker, which will be elected by the ZooKeeper.

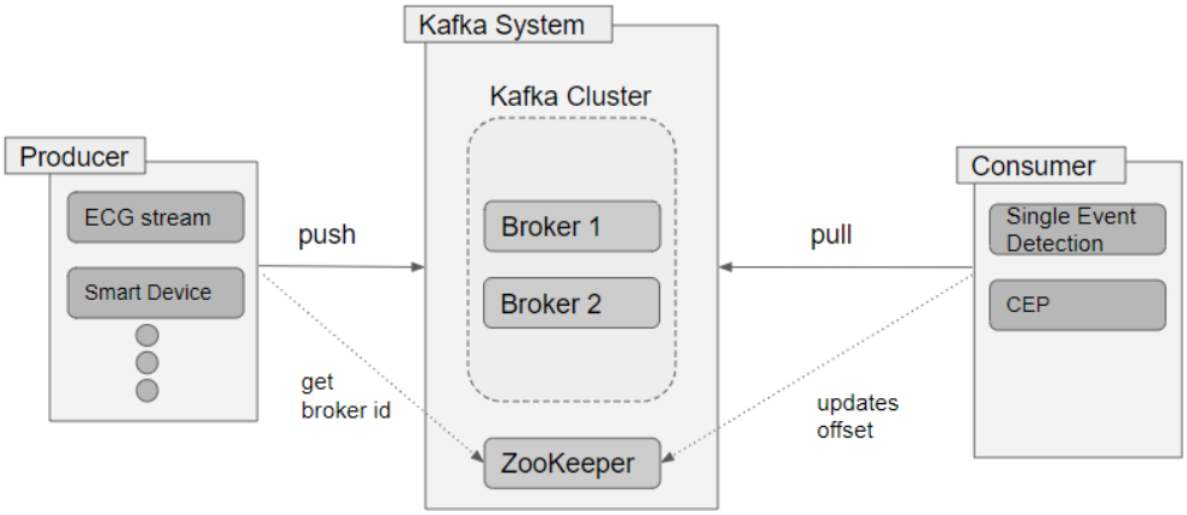

Figure 5.2: Overview of Kafka architecture for this case study. Illustrating the producer, cluster and consumer architecture

Kafka ZooKeeper: The ZooKeeper is the central management and coordination unit in the cluster. If a new broker enters the cluster or if a broker fails, the ZooKeeper notifies the producers and consumers. This notification allows producer and consumer on how to act and start communicating with other available brokers in the cluster.

Kafka Producers: Producers push the data to the brokers. The Kafka Producer passes data to partitions in the Kafka topic based on the partition strategy that has been predefined. The Kafka producer takes care that the data rate can be handled by the broker.

Kafka Consumers: The Kafka brokers are stateless, while the consumers maintain the number of messages that have been consumed. This is achieved by using a partition offset. A consumer confirms each message offset which is proof that all messages before were consumed.

5.2.1.2 Kafka-Concepts

Kafka Topics: A Kafka topic is a logical message channel to distinguish between different message types. Kafka topics are divided into several partitions. Partitions allow parallelizing a topic by splitting the data into a specific topic across multiple brokers. A partition writing process of a topic is illustrated in figure 5.3. Each partition can be assigned on a separate machine to allow multiple consumers to read from a topic in

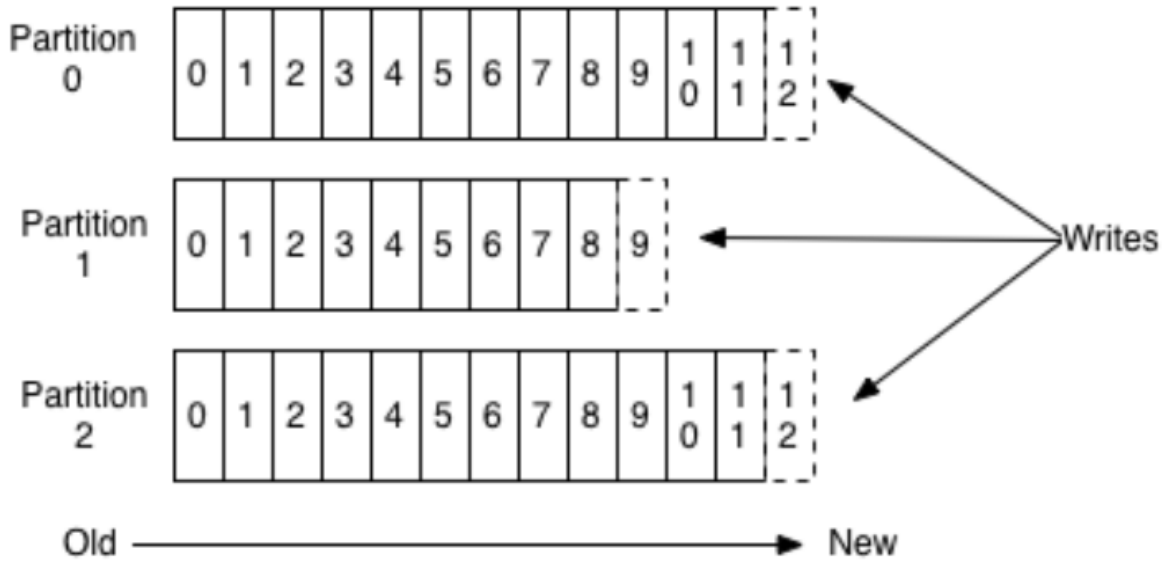

Figure 5.3: Showcase of a kafka topic with 3 partitions. New entries are written with the offset number in the top of a queue

parallel. Further, can the message processing throughput of a consumer be increased by the parallel reading of multiple partitions.

Each message within a partition has the previously mentioned offset. The offset represents the ordering of messages as an immutable sequence. The consumer has different options to start reading from any offset point, which allows the consumers to join the cluster at any time.

With the offset info, each specific message in a Kafka cluster can be uniquely identified with the information of the message topic, partition and offset within the partition.

Partitions in Kafka: Each broker holds several partitions that can be either a leader or a replica for a topic. A leader coordinates all writes and reads to a topic, and coordinates updates in the replicas with new data. If a leader fails, a replica takes over as the new leader.

Topic Replication Factor: Topic replication helps to prevent system failure in case of broker failure. In case of a broker failure, the topic replicas of other brokers can secure the system stability. A replication factor determines how many backups are created for a topic.

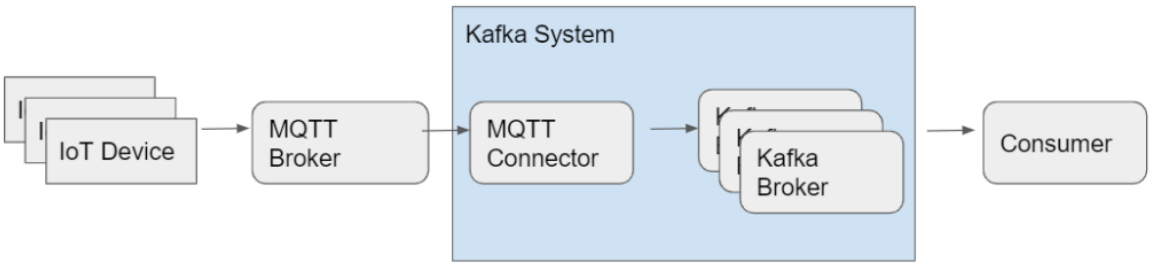

Figure 5.4: MQTT Broker ingestion

5.2.2 Communication protocol between producer devices and the Kafka ingestion system - MQTT

MQTT is an ISO messaging protocol [MQT]. MQTT is famously used for IoT scenarios and uses a publish/subscribe architecture in contrast to HTTP with its request/response paradigm. The publish and subscribe concept is an event-driven paradigm and allows messages to be pushed to a client. The central communication point is an MQTT broker. A broker dispatches the messages between the senders and consumers. Clients publish the messages specified on a topic to a broker. This provides bi-directional communication between the devices over an MQTT broker.

MQTT and Apache Kafka are a good combination of end-to-end IoT integration from the edge to the processing nodes and back. In this case-study, the bidirectional and inter-device communication was not needed for the goal of having a monitoring and processing pipeline along with the IoT architecture with a centralized deep analytics engine. Nevertheless, the MQTT characteristics of being bandwidth and battery-efficient on IoT devices are considered a valuable component for this architectural concept. By evaluating different implementation recommendations of MQTT device integrations, the following two integration scenarios were evaluated.

Scenario 1 - MQTT Broker:

In this first scenario (figure 5.4), the data gets pulled from the MQTT Broker via the Kafka MQTT Connector to the Kafka Broker.

Scenario 2 - MQTT Proxy:

To reduce complexity and simplify the management of the infrastructure, in Scenario 2 (figure 5.5), an MQTT proxy can be used to get rid of the MQTT broker cluster that would be required to be managed in Scenario 1. In case, no additional broker clus-

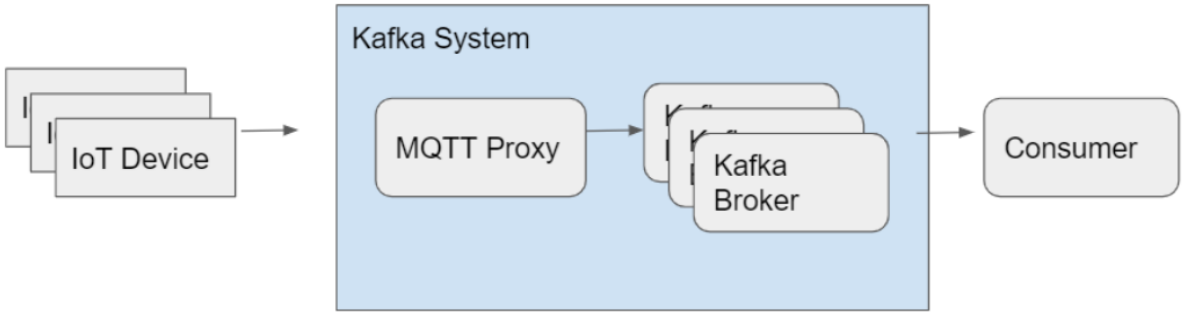

Figure 5.5: MQTT Proxy ingestion

ter is needed, more reliable and performant processing can be guaranteed by having fewer components along the processing pipeline.

5.2.3 Stream Processing and single-stream event detection - Faust

Faust is a high performance stream processing library, which adapts the idea of Kafka Streams in Python and for this chosen for the case-study. Faust has the following characteristics:

1. Kafka Consumer: Faust worker can act as a consumer of a Kafka cluster and process events based on their assigned topics.

2. Scalability: Faust can be vertically scaled by assigning multiple workers to a topic.

3. No DSL: It is written in python. The whole range of python libraries is available for processing an event, which includes, e.g., Tensorflow for more sophisticated AI-based analytics.

4. Asynchronicity: With Python 3 and the *AsyncIO* module, Faust offers high-performance asynchronous processing. With *AsyncIO* and the *async/await* keywords in Python 3.6+, multiple stream processors can be run in the same process.

5. Stateful: Faust can persist in states and act as a database. Tables are named distributed key/value stores which are implemented as regular Python dictionary.

6. Stream Windows Processing: Tumbling, hopping and sliding windows are available. Windows can be subject to time constraints in which old windows can expire to stop loading data.

```python
@app.agent()
async def myagent(stream):
    async for event in stream:
        ...  # process event
```

Figure 5.6: Faust agent basic code example

7. High Reliability: For reliability, Kafka topics possess a write-ahead-log. In case a key is changed, the information gets published to the changelog. Standby nodes consume the changelog info to keep a replica of the data. In the case of a node failure, this replica enables an instant recovery.

8. Persistence: Tables can be stored locally on each machine using RocksDB [Roc], a high performance embedded database for key-value data.

   The core components are described in the next 5 paragraphs.

Agents: Agents process infinite streams using asynchronous generators. The agent concept comes from the actor model, where the stream processor executes concurrently on many CPU cores. An actual code example can be seen in Figure 5.6

Tables: Tables are dictionaries that give stream processors states with persistent data. Sharding and partitioning are an essential part of stateful stream processing and need to be planned strategically for the right processing in a Faust cluster. Streams can be processed in a round-robin principle where Faust acts as a simple event processing and as a task queue.

Distribution: Faust relies on Kafka's consumer management principle, to identify whether any partition or topic is not being served to and launches an agent in an instance where the application is reporting as functioning.

Figure 5.7 represents a single Kafka topic that has six partitions. Three Faust worker are running on the same application name. The left illustration shows a working cluster with an elected leader and two further worker nodes. The elected topic leader will coordinate which partition will be fed to which worker. The node failure detection is based on a timeout value, after which a consumer group will inform the leader that one of the nodes is down and trigger a re-distribution of topics and partition assignments.

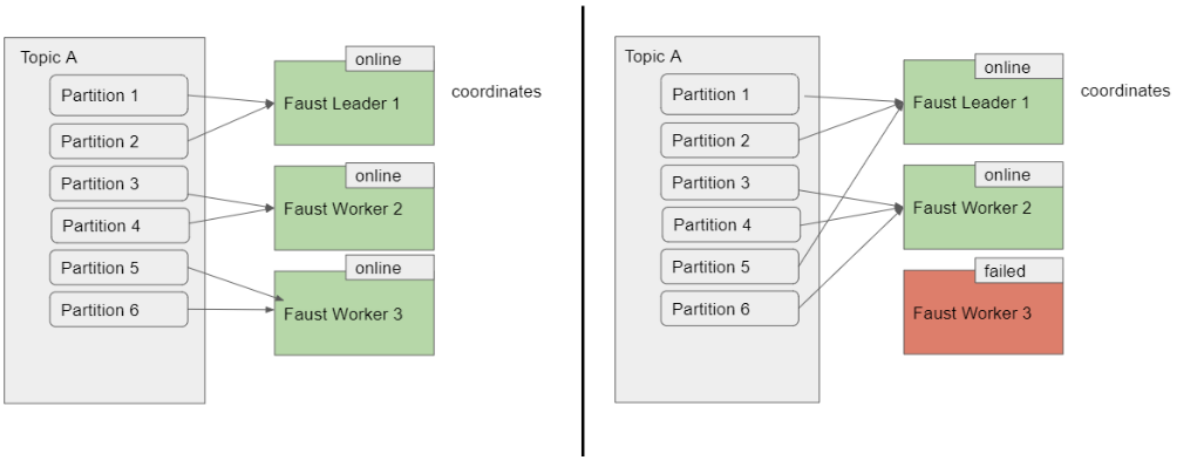

Figure 5.7: Faust topic replication example in case of a worker failure. Left side - All worker healthy; Right side - one worker fails and partitions are redistributed

Table Sharding: Tables shards in Kafka are organized using a disjoint distribution of keys. Any computation for a subset of keys happens together in the same worker. The following example illustrates an incorrect usage of key subsets assignments of worker processes: If we have patient and location information, depending on the use case and goal, we want to partition by a key on location or patient-ID. Should we partition on a patient-ID and process aggregated information on location information, most likely we run into a risk of incomplete data, where location information might be spread across different partitions. Such an issue can be solved with Faust by using two distinct agents and repartition the stream by location when populating the table.

The Changelog: The changelog is used to recover from system failure, were every modification to a table has a changelog entry. The changelog is registered in the Kafka Cluster as its topic and only keeps the most recent values of a key in the log. The RocksDB storage allows the recovery of tables, where a worker retrieves missed updates since the last time the instance was running.

5.2.4 Complex Event Processing - Kafka Streams with KSQL

Faust acts in this Kafka infrastructure as a consumer for processing the data streams as well as producer of event streams. With this architecture design, CEP Engines can now be attached to the Kafka cluster as a consumer and can access those event streams in the Kafka architecture. In the implementation of the case study, the focus will be held on the “simple” CEP integration. This integration will not have the

benefit of integrating complex reasoning over ontologies but explains the composite event behavior reasoning. At the end of this chapter, a conclusion over the advanced possibility of integrating a semantic CEP in this use case will be given.

Based on the underlying Kafka ecosystem, the decision was made in favor of Kafka Streams. Kafka Streams is the Apache Kafka library for writing streaming applications and microservices in Java and Scala.

5.2.4.1 Kafka Stream Concept

Kafka Streams has following characteristics [Kafb]:

1. Lightweight client library, that can be embedded in Java application.
2. Kafka is the only dependency as an internal messaging layer. Kafka Streams uses Kafka's partitioning principles to scale while maintaining ordering guarantees horizontally.
3. Fault-tolerant local state which enables stateful operations (windowed joins and aggregations).
4. Exactly-once processing is guaranteed, where each record will be processed once, even in case of a failure.
5. One-record-at-a-time processing for lower processing latency.
6. Event-time based windowing operations.
7. Includes stream processing primitives with a high-level DSL and low-level Processor API.

The core components are described in the next paragraphs.

Kafka Stream Processing Topology: A stream processing application defines its logics trough one or more processor topologies. A processor topology represents a graph of processor nodes that are connected by their streams. A stream processor represents a processing step of a stream input and applies operations to produce one or more outputs, which are sent to downstream processors in the topology.

There are two type of processors in the topology:

1. Source processor: A source processor produces an input stream from one or multiple Kafka topics and acts as a consumer of topics to forward them to down-stream processors.

2. Sink processor: A sink processor does not have down-stream processors. It received records from its up-stream processors and sends the output to a specified Kafka topic.

   Processed results can forwarded to Kafka or to external systems.

Time Dimension: Common notions of time in streams are:

- **Event-time**: When an event or data record occurred. In this case study, e.g., the record of the ECG signal at the device.
- **Processing-time**: When the event or data record is being processed. This can be the time of consumption and later than the event time.
- **Ingestion time**: When the event or message is stored inside a topic partition by the Kafka broker.

Kafka automatically embeds the timestamps into a message, where timestamps can represent event-time or ingestion-time. The corresponding Kafka configuration setting can be specified on the broker level or per topic.

Aggregation: Aggregations take one input stream or also table and create a new table by combining multiple records into a single output. An example is the computation of a sum.

Windowing States: Kafka Streams offers windowing and lets group records that have the same key for stateful operations such as aggregations or joins. Windows can be specified with a retention period and are tracked per record key. Kafka streams state management enables joins over input streams, grouping and aggregation over records.

Processing guarantee and out-of-order-handling: Stream processing applications usually make use of a lambda architecture, which enables the infrastructure for real-time and batch processing in parallel. The goal is often to have real-time alerts and a

batch layer that guarantees that no data-loss or data duplicates occur. Newer Kafka versions allow its producers to send messages to different topic partitions in a transactional and idempotent manner. With this characteristic, Kafka Streams offers the end-to-end exactly-once processing semantics without the need of adding a batch layer[Kafa]. For stateless operations, out-of-order data will not impact the processing logics since only one record is considered at a time for stateful operations such as aggregations and joins. For the handling of out-of-order data, a longer wait time should be considered, where the state needs to be tracked. This will lead to a trade-off between latency, cost, and correctness.

Some of the out-of-order data cannot be handled by increasing the latency and cost in Streams:

1. Stream-Stream joins: All three types (inner, outer, left) handle out-of-order records correctly. The only problem can occur in the resulting stream, where left and outer joins contain unnecessary 1: null records.

2. Stream-Table joins: Out-of-order records are not handled and may result in unpredictable results.

5.2.4.2 Kafka Stream Architecture

Stream Partitions and Tasks: Kafka partitions data for storing and transporting, while Kafka streams partitions for further processing. In Kafka, partitioning enables data locality, scalability, high performance, and fault tolerance. With partitions and tasks as a logical unit, a parallel processing model can be realized.

Following similarity can be found between Kafka streams and Kafka:

- A stream partition is an ordered sequence of data records assigned to a Kafka topic partition.

- The keys of a topic element determine the data partitioning in Kafka and Kafka Streams.

The processor topology can be scaled by using multiple tasks, where a task is a fixed unit of parallelism for a topic. The instantiation of a processor topology is done by a task, where the task also maintains a buffer for each of the assigned partitions and process messages one-at-a-time from the record buffers. This characteristic gives the Kafka stream architecture the independent and parallel processing paradigm.

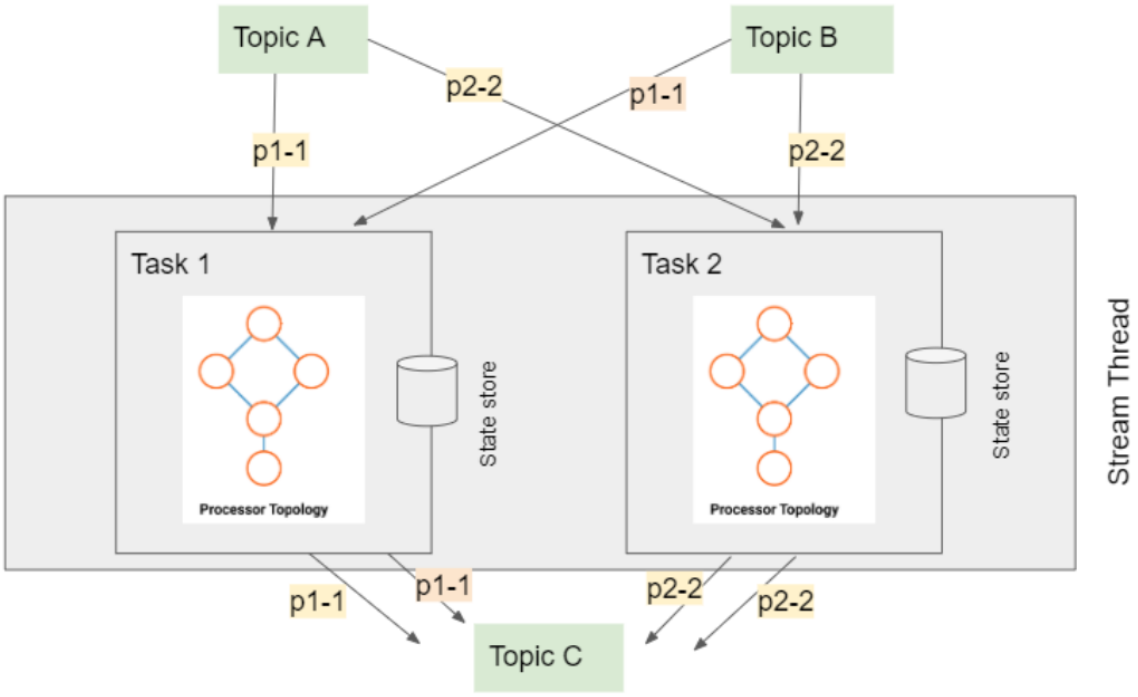

Figure 5.8: Illustration of a task assignment inside a Kafka Stream thread.

Kafka Streams can be run on a single machine or spread across multiple machines, where tasks are automatically distributed to the application instances. The diagram in Figure 5.8 exemplifies how a task assignment structure could look.

Two tasks, where each one is assigned with one partition of the relevant topic, which in return enables a fault tolerance characteristic for processing. In case an application instance fails, all affected partitions will be automatically restarted on another instance and fed with the same stream partitions.

Threads While the partitions, in combination with a task, enable fault tolerance, the threads enable parallel processing inside an application instance. With creating more threads, a different subset of partitions is used to enable parallelized processing. The threads run independently and can not communicate with each other. Kafka Streams takes care of distributing the partitions among the tasks.

Local State Store State stores enable to store and query data inside a task, which is essential for stateful operations (e.g., aggregation over a time window or join). The state stores are secured of failure by a changelog topic with each update, similar to the changelog of Faust. A changelog topic is partitioned so that each task state store has its partition.

5.2.4.3 KSQL

KSQL is built on Kafka Streams as a streaming SQL engine. Based on the simplicity of use and high expressiveness, KSQL was used as the (semi-)CEP for detecting composite stream events in this work.

5.3 Technical design: Data Processing - ECG Data

For the part of processing the ECG data along the stream pipeline, the following considerations were evaluated:

1. Where to process data and how does it scale: If the data processing step has the output result of reducing the overhead over the eco system, the goal should be to put that processing step as close as possible to the source. Possible limitations in such a scenario can be the computing power or memory resources on the IoT device or edge nodes. Reliability could be affected by putting a to high and fluctuating load on devices, which could lead to an overload in the processing and result in processing problems. In some scenarios, the battery life optimization is critical, where the processing power should be kept as low as possible.

2. Some processing steps might be based on semantically enriched data inference, which contains the linking of static data that can be subject to specific data protection laws. This can have as a result that personal data cannot be shared with end devices or end device data can not be shared with a cloud system.

3. In a scalable processing architecture with parallel worker nodes, stream elements could be processed on different worker nodes, and time-series correlation could be lost. A possible solution is to assign data streams on dedicated worker nodes. Another way is to structure the processing in a way that it will be decoupled from the time-series.

4. Limitations on message size along the processing pipeline. For example, MQTT has the limitations of the length of the actual topic string at most 65536 bytes, and the payload of the message is limited to 268,435,456 bytes [MQT]. It needs to be evaluated that the message processing can be handled at later processing nodes of the other systems like Kafka and Faust, or a reduction of the data size considered.

5.3.1 Pre-Processing

As an initial edge pre-processing step, the standardization and normalization of the ECG data stream was conducted. Only standardized data was feed into the Kafka message broker and guaranteed consumers that the data contains identical data and semantic structure. The heterogeneity of IoT devices is a primary challenge in IoT systems. Different ECG devices can be clocked on different frequencies. The initial step was to sample the raw data streams on 180-hertz frequency, which contains downsampling in case of a higher frequency and up-sampling in case of a lower frequency. The high-frequency noise filter is the second computation step to retrieve a smooth ECG stream. As a final processing step, the data has been normalized on a scale of 0-1. The resulting standardized data stream can now be processed in the pipeline with different heterogeneous IoT devices in the case of ECG monitoring.

5.3.2 Core-Processing

The HTM algorithm was used to find anomalous segements on the standardized data stream (explanation in section 5.4.1). Anomalous segments are forwarded for storage inside the Kafka cluster for further processing by consumers. Topic-specific worker nodes will now be able to process the anomalous segments.

From the anomalous segment to the extracted signal A single extracted signal is relevant for the classification in most of the methods. The signal extraction in this thesis was based on the following step-wise approach:

1. Splitting the continuous ECG signal to 5s windows and select a 5s window from an ECG signal.

2. Amplitude normalization between the range between 0 and 1.

3. Determine the set of local maximas based on zero-crossings of the first derivative.

4. Determine the set of R-peaks candidates by applying a threshold of 0.9 on the normalized value of the local maximas. Experiments showed that the threshold can be in between 0.85-0.92 for the R-peak detection.

5. Determine the median of R-R time intervals as the nominal heartbeat period of that window (T).

6. For each R-peak, selecting a signal part with the length equal to 1.2T. With this value, it should be assured that all beats are completely extracted.

7. Padding each selected part with zeros to m.

8. Send signal for classification.

The pseudo-code of the signal extract can be found in appendix 7.3. This approach is a very basic methods to extract signals. More advanced methods are already proposed by other researchers and were not in the scope of this work, but will be discussed in the conclusion.

5.4 Technical design: Anomaly detection, classification and complex event processing

5.4.1 Anomaly Detection

ECG monitoring can be classified in the applications of short term and long term monitoring. An ECG signal in a long term monitoring scenario will create data with a frequency of 360 hertz on a two signal stream with timestamp, resulting in 632 MB/day. An ECG signal stream usually contains a high amount of data that is considered normal. Periodical deviations are considered anomalous signals. Those signals need to be analyzed and classified for specific diagnoses. Based on these characteristics, the next processing step is to identify and extract those anomalous signals.

Following the idea of an edge event detection, that contains the classification of ECG data, an implementation was tested to detect close to the source anomalous signals, extract them and send them in the Kafka infrastructure. This has the benefit of reducing the load on the communication ports and Kafka cluster. A modern approach for online detection of anomalies in time series data presents the HTM algorithm.

Hierarchical temporal memory: HTM was inspired by the learning behavior of the neocortex, where the signals from all of the body's sensory organs are processed in one algorithm [Mou78].

HTM is a self-learning and memory system of storing invariant representations of physical structures and abstract concepts. Self-learning is described as the ability that no offline data set for pre-training is needed and the learning is done in real-time on the

incoming data. HTM is highly adaptive and can learn changing representations in real-time and forgets outdated representations. This allows HTM to learn with invariant representations of common patterns. By observing those patterns in a sequence, HTM can predict future patterns.

HTM structure: The HTM learning algorithm models how learning takes place in a single layer of the cortex. HTM creates sparse representations of time series patterns. When the HTM algorithm observes a new pattern, it will try to match it to stored patterns. Because inputs never repeat in the same way, invariance of the stored sequences is vital to the ability to recognize inputs. The HTM algorithm only knows what patterns are probably following a particular observed pattern. HTM's general learning principles are to train on every input. If a pattern is repeated, then strengthen it and if the pattern is not repeated, then forget it. HTM carries out at each time step three steps on the observed input, which is further explained in the appendix 7.1.

For each sample, an anomaly score gets determined. In a sliding window over the ECG stream, the moving average of the anomaly score will be determined and based on a patient-specific anomaly threshold. Two functions can be triggered by the HTM result:

- Early warning signal for a local caretaker on IoT devices or edge nodes like a raspberry pi.
- Forward the segment to the Kafka infrastructure, where it will be ingested for further in-depth processing over the anomalous segment.

The threshold can be configured depending on the clinical risk of the patient. The threshold represents the trade-off between detection accuracy, data rate, and with this the load on the ecosystem. To catch all anomalous signals, a very sensitive threshold needs to be selected, which in return increases the amount of data that is classified as anomalous. A high threshold will reduce the data load on extreme cases only, that can lead to missing unhealthy beats. The pseudo-code for the HTM algorithm implementation can be found in Appendix 7.2.

5.4.2 Classification

In the domain of ECG classification three main approaches have been identified:

1. **Feature-based**: It aims to find a set of relevant features of ECG data that can attain a good classification accuracy.

   In ECG signals (see Figure 5.9), one cardiac cycle includes the P-QRS-T waves. This feature extraction technique determines the amplitudes and intervals in the ECG signal for subsequent analysis.

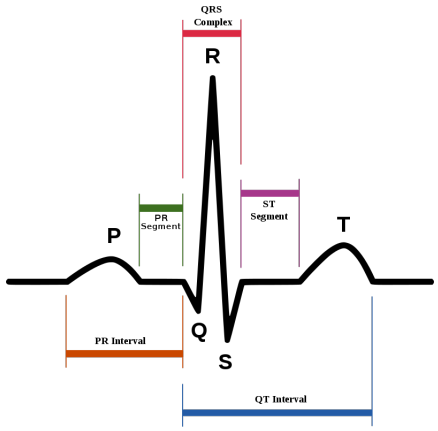

Figure 5.9: Example visualization of an ECG signal and its feature definitions [wik].

2. **1-Dimensional ECG signal classification based on CNN**: An adaptive implementation of 1-D convolutional neural networks (CNNs) is used to fuse the two significant blocks of the ECG classification into a single learning body: feature extraction and classification.

3. **2-dimensional ECG image classification based on CNN**: One-dimensional ECG signals can be transformed into two-dimensional ECG images, where noise filter and feature extraction are not required. Using ECG image as an input also benefits in robustness. Traditional ECG arrhythmia detection methods are sensitive to noise. However, when the ECG signal is converted to the two-dimensional image, the proposed CNN model can automatically ignore the noise data while extracting the relevant feature map throughout the convolutional and pooling layer. Thus, the proposed CNN model can be applied to the ECG signals from the various ECG devices with different sampling rates and amplitudes. Furthermore, detecting ECG arrhythmia with ECG images resembles how medical ex-

perts diagnose arrhythmia since they observe an ECG graph from the patients throughout the monitor, which shows a series of ECG images [JNK+18].

To keep the complexity of the classification as low as possible, the 1D CNN classification was considered, while a future improvement should be considered by the current state of the art 2D CNN approach.

Model Training: The MIT-BIH Arrhythmia Database [MM01] was used for training the model, containing 48 analysis two-channel 30 minutes ECG recordings, obtained from 47 subjects. The original recordings were digitized on a frequency of 360 per channel with an 11-bit resolution over a 10 mV range. The focus of this work was on the optimization of the CNN model.

5.4.3 Result Evaluation

This section studies the effects of applying the HTM algorithm as a pre-filter step to identify anomalous segments for further analysis by the Faust classification cluster. The study will focus on the trade-off factors of the implementation configuration over different threshold values. A second evaluation was conducted on the the Faust worker extraction and classification task, where time measurements were taken for different arrival rates of incoming anomalous segments. The final part of the result evaluation shows the total troughput time of the signal sample generation till the finished classification of the Faust worker.

5.4.3.1 Tradeoff 1: HTM anomaly threshold vs. data size vs. detection rate

The evaluation was conducted on the labeled MIT-BIH Arrhythmia Database [MIT]. This data set contains approximately 110,000 beats that have been annotated by two or more cardiologists independently. Each patient has a recording of 30 minutes in a frequency of 360 Hz, which results in 648,000 samples. The annotations were mapped by the cardiologist to a single sample, were an example can be seen in Figure 5.10:

The HTM algorithm, as well as the implementation, contains a complex setup of parameters. The fine-tuning of the HTM was outside of the scope of this thesis. The following parameters are used in the experimental setup and can be seen in Figure

Time	Sample	Class
...		
0:05.025	1809	N
0:05.678	2044	A
0:06.672	2402	N
...		

Figure 5.10: Snipped of the ECG annotation set. The first column marks the time of a signal. The class expresses if the signal was marked as healthy or unhealthy.

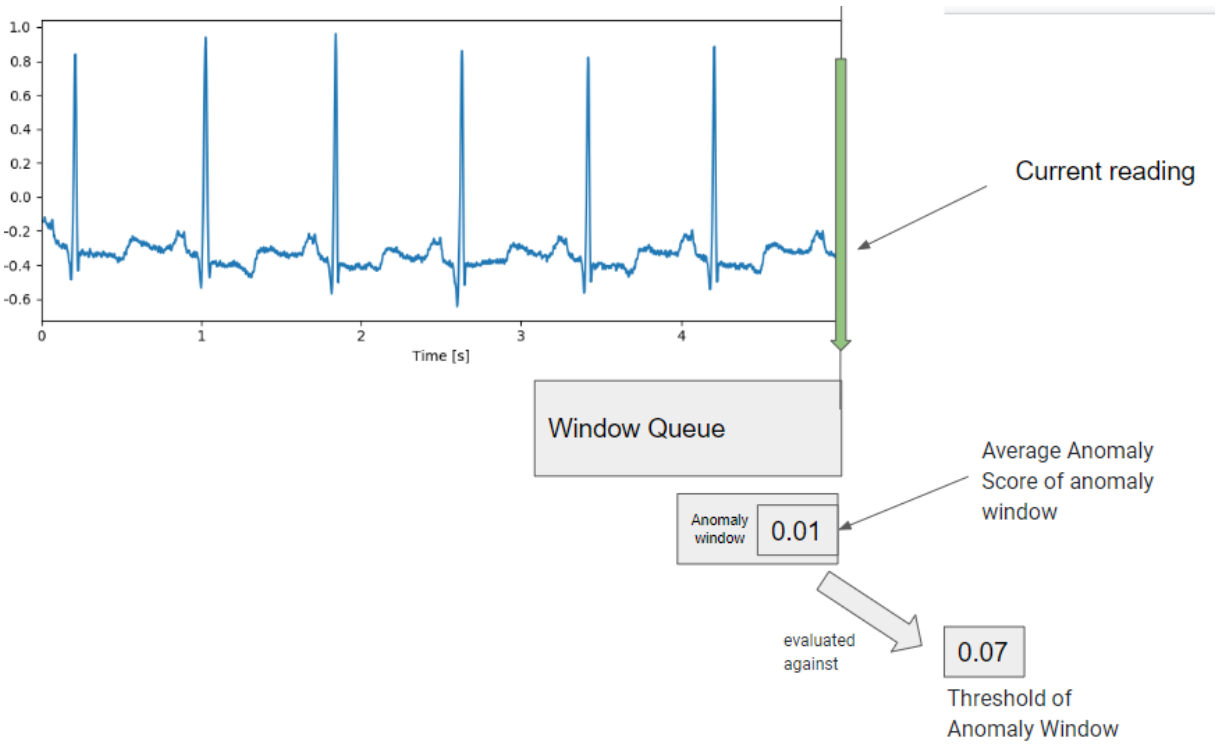

Figure 5.11: This is a model of the implementation parameter used for identifying and extracting anomalous segments with the HTM algorithm

5.11:

- Maximum and Minimum queue window size = Maximum size of a window.
- Hopping window size = Represents the overlapping interval.
- Anomaly window size = Length of the moving average of the anomaly score.
- Threshold of moving average of anomalies = The factor of tolerance over detected anomalous segments.

In this experiment, the frequency was downsampled to 180 Hz to reduce the experiment duration (for different frequencies of an ECG stream, the implementation

parameters need to be optimized and reconfigured). The raw data stream was not pre-processed and contains noisy samples.

For the evaluation on a 180hz frequency, following settings were choosen:

- Maximum and Minimum queue window size: 200-450
- Anomaly window size: 20
- Hopping window size: 100

The major goal of the HTM algorithm was to detect unhealty signals and reduce as good as possible the data rate on the infrastructure. For this, the detection rate of an unhealthy signal (True Positive Rate = TPR) was evaluated, as well as the rate of healthy signals that were considered as anomalous (False Positive Rate = FPR). The results can be seen in visualized in Figure 5.12. With a low threshold of 0.06, the TPR was at 100%, while 85% of healthy signals have been considered for the ingestion in Kafka and the precise classification by the CNN model. With a threshold of 0.08, it can be observed that the TPR is at 94%, while the FPR shrinks to 65%.

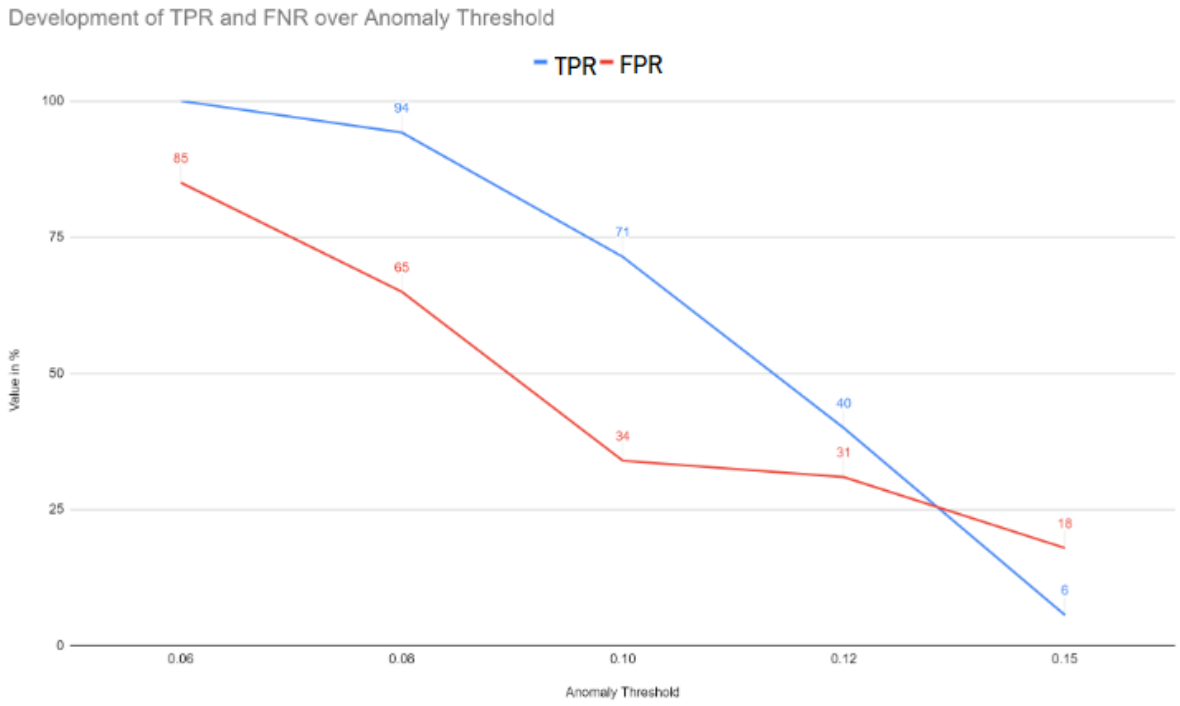

Figure 5.12: Evaluation of Development of TPR and FPR over different anomaly thresholds.

For the same setup and threshold ranges, the data volume was evaluated and can be seen in figure 5.14. The initially measured generated data of one ECG sensor over

24 hours is at 480MB. With an acceptable anomaly threshold of 0.08, the volume was reduced by 24% to a volume of 355MB. All results of the threshold test can be viewed in the table 5.13.

Considering that the HTM algorithm had no optimization of the hyperparameters and the raw noisy data stream, were the results promisingly good. The HTM algorithm should be in a future outlook investigated on a pre-processed and clean ECG stream (e.g, noise filter, feature extraction, representation schemas).

Anomaly Threshold	TPR (recall) in %	FPR (fall-out) in %	Precision in %	FPR (fall-out) in %	Volume reduction in %	MB/24h	TP	TN	FP	FN
0.00	100	100	1,5	100	0	480	35	0	2239	0
0.06	100	85	1,8	85	15	408	35	336	1903	0
0.08	94	65	2,2	65	26	355	33	784	1455	2
0.10	71	34	3,2	34	40	288	25	1478	761	10
0.12	40	31	2,0	31	80	96	14	1545	694	21
0.15	6	18	0,5	18	86	67	2	1836	403	33

Figure 5.13: Evaluation table of binary classification analysis of untuned HTM anomaly detection over the raw data stream of patient-100.

The processing was evaluated on an AWS instance type t2.medium with 2 vCPUs (3.3 GHz) and 4 GiB RAM.

Processing time (highly depending on sample frequency and was tested with 180hz):

- HTM algorithm: $\leq$ 0.01 seconds per sample.
- Queue segment evaluation: $\leq$ 2 seconds (mainly depending on window size configuration).

5.4.3.2 Faust signal extraction and classification evaluation

In this experiment, the extraction and classification of anomalous signals were evaluated on a Faust worker node. A single Faust worker was deployed where the test consisted of measuring the time of extracting a signal and classifying it. The Figure 5.15 shows the evaluation results, where different event rate scenarios where tested. The worker was able to handle 245 events perseconds and showed a linear behavior for increased data rates. The experiment showed that the classification step took slightly more time than the signal extraction process.

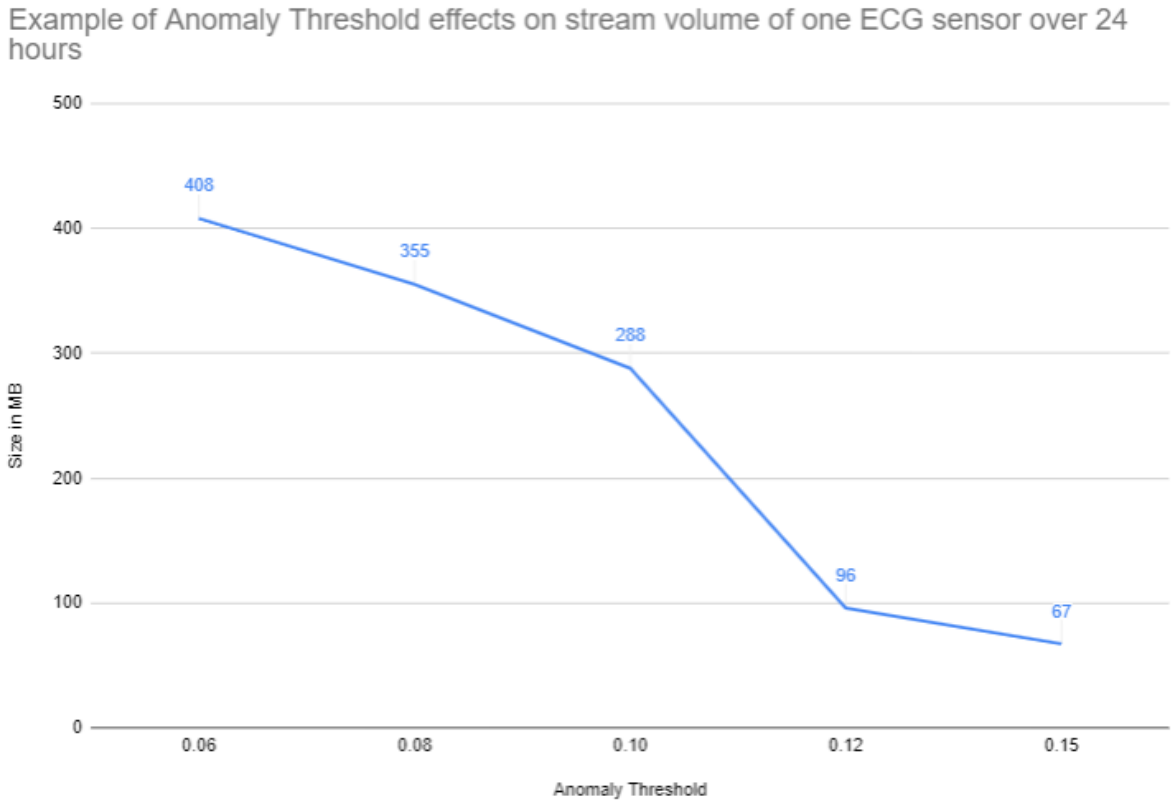

Figure 5.14: Example of Anomaly Threshold effects on stream volume of one ECG sensor over a 24 hours duration, where a raw generated volume was estimated with 480 MB every 24hours.

5.4.3.3 Processing time along the Architecture

This section contains the throughput time of a generated event at the sensor until the final event classification at the Faust worker node. Several measurements were taken at the HTM algorithm, MQTT connection, Kafka cluster, and Faust worker node, which can be seen in figure 5.16.

The HTM algorithm processed the raw ECG samples on a frequency of 180Hz, where the analysis time was between 0.4 and 2 seconds of a single signal, which is due to the window operator configuration. Depending on the window size, the processing time will be affected. The HTM algorithm gives an anomaly score on a single signal in a fraction of a second. To establish a mechanics to prevent a system flooding based on noise elements inside the raw data stream, the evaluation was delayed untill an clear decision can be made based on the followup samples of the time series of an ECG.

The network communication latency between the IoT device, Kafka ecosystem and Faust worker node are essential measurements to be considered for a real appli-

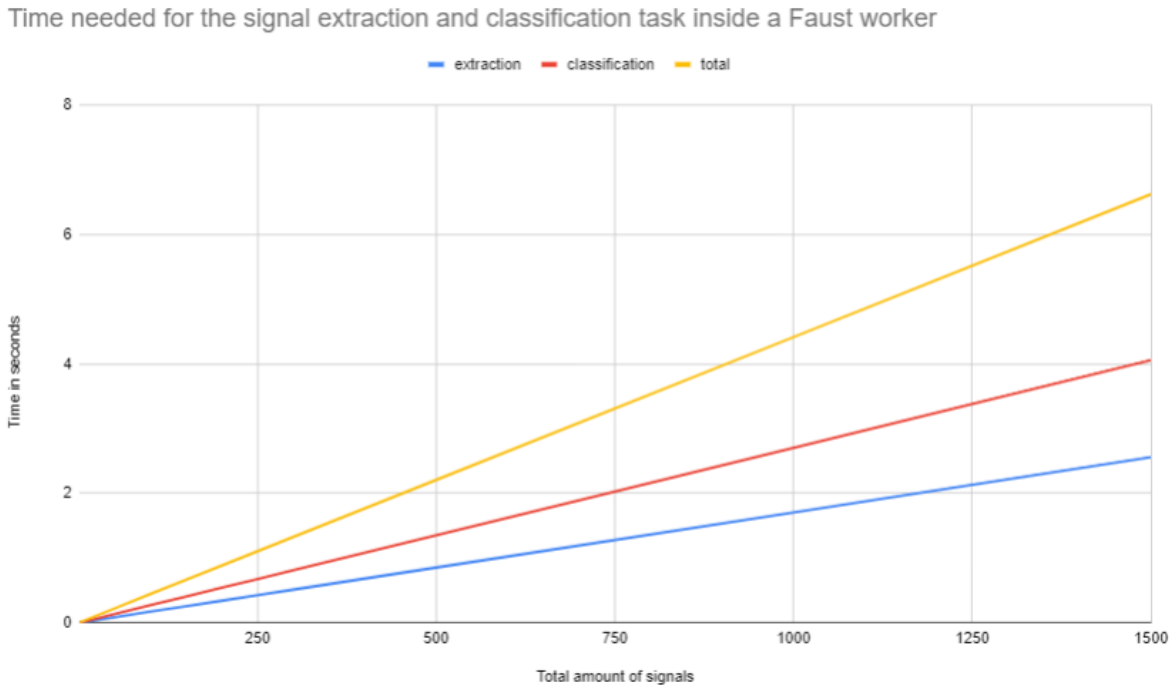

Figure 5.15: Processing time of a Faust worker over anomalous segments.The task consisted of signal extraction and classification.

cation. In this case study they were not considered for an in-depth investigation and outside of the scope. Basic measurements were taken and showed a communication time of under 0.2 seconds of a sensor deployment in Barcelona to the AWS availability zone 3 in Frankfurt.

Two measurements were taken at the Faust worker node for the anomalous segment task to extract signals and classifying. The total time of a segment classification was at under 0.006 seconds (note: an anomalous segment was consisting of 3 signals in the test based on the anomalous segment window configuration).

The maximum throughput time starting at the generation of an event untill the classification of probable unhealthy signals was at 2.1 seconds plus the network latency characteristics in the deployed environment for MQTT and between the Kafka and Faust worker.

5.4.3.4 Classification

The model was implemented in a basic 1D-CNN to test the overall functionality. The signal pre-processing step follows the idea of [JNK+18]. By having the same input structure, the model can be replaced with the model of Jun et al. proposed 2D-CNN

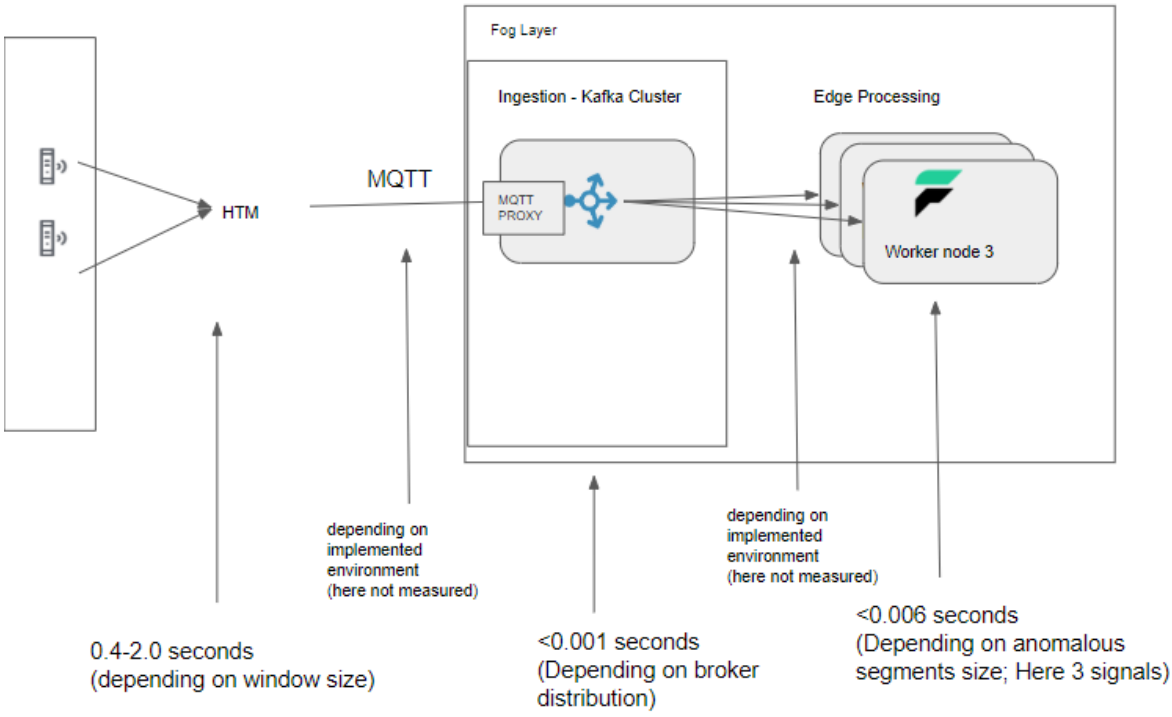

Figure 5.16: Visualization of the total throughput time of one ECG signal.

model architecture, that showed the following result for the classification of the same ECG data with the same setup [JNK+18]:

- Acc(%)= 99.05
- Sp(%) = 99.57
- Se(%) = 97.85
- +P(%) = 98.55

In a real application, the implemented model needs to be replaced, whereas the remaining data pipeline can stay unchanged..

5.4.4 Complex event processing

CEP can be used to predict and manage critical assistance. A CEP system may combine the processing of several data streams with static data stored in a database. Possible use cases can be countless, depending on the available data. Some use cases are:

- Realtime (remote-) Reasoning and alert system for healthcare personnel about a patient.

- Feedback loop with devices that monitor the patient health, for (automatic-) prescriptive treatment and medication administration.
- Information system for caregivers and relatives about long term health development analysis (e.g., lack of movement, pro-active-emergency detection).
- Realtime Linked data correlation with open health and community statistics, to detect, for example, cancer clusters or spikes in asthma attacks in a community.
- Linking of data sources and processes to automate the tasks assignment and process allocation of resources to teams during a disaster management [XAB+11].

5.4.4.1 Event Processing Network Diagram

In this section, the data event topics and channels are visualized (see Figure 5.17). The event data flows are described as channels, which are the streams that effectively handle the routing of events. Event processing begins with raw event data from the medical devices and then ends on the right side, where complex event alerts are generated.

The intermediate queries are shown below for each event processing step. This implementation is just a showcase and in a real application, measurements and thresholds must be evaluated.

(1) Average HeartRate Query

```
SELECT
AVG(hr) As AverageHr
FROM
hrTopic[ROWS4 SLIDE4]
```

(2) HeartRate History Query

```
SELECT averageHr AS averageHr
FROM averageHrTopic[ROWS 4]
```

(3) HeartRate Difference Query

```
SELECT
a.averageHr AS currentAverageHr,
h.averageHr AS historicAverageHr,
a.averageHr - h.averageHr AS hrDifference,
((a.averageHr-h.averageHr)/h.averageHr)*100 AS hrChangePercentage
FROM
averageHrTopic[now] AS a,
historicAverageHrTopic[ROWS 1] AS h
WHERE
a.averageHr>0 AND h.averageHr>0
```

(4) HeartRate Alert Query

```
SELECT
|'Hr increase of'||hrChangePercentage||'from'||
averageHr||'to'||  hr AS Alert
FROM
hrDifferenceTopic[now] AS d
WHERE
d.hrChangePercentage>5
```

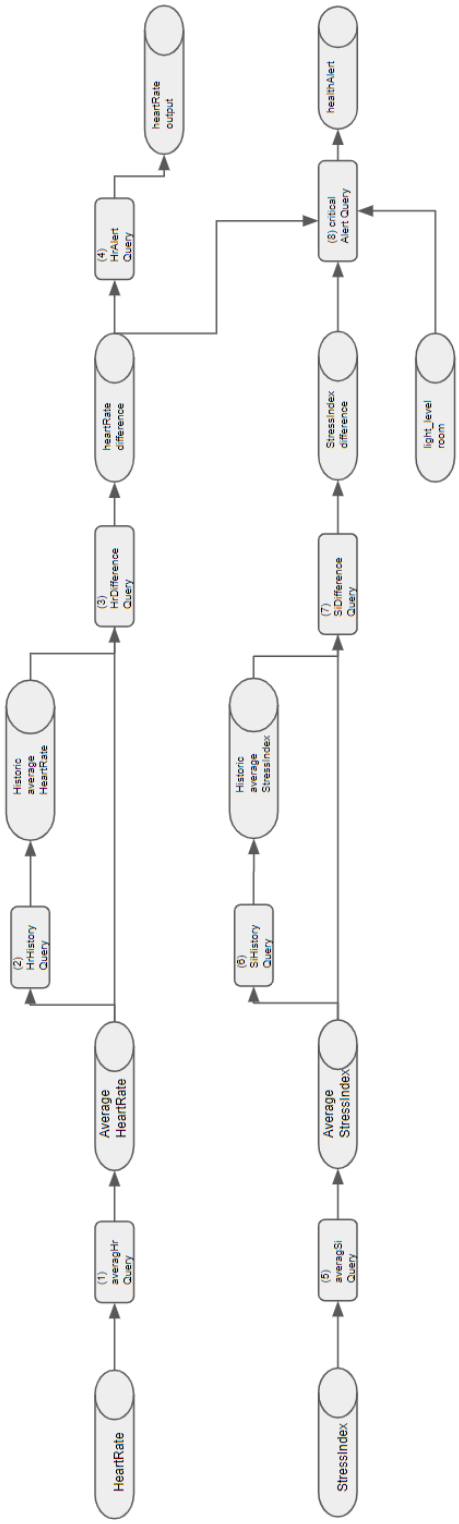

Figure 5.17: Event Processing Network Diagram of a complex event scenario for critical health status detection. The example contains the correlation of the heartrate and stress measurement event composition.

5.5 Data Representation and Enrichment

IoT semantic interoperability The representation will be a fusion of already existing healthcare ontologies and IoT sensor stream representations. The existing healthcare ontologies will be not defined in this work, and the focus held on the IoT stream representation and the possible option of linking the data. The IoT stream ontology decision factor will be the trade-off between expressiveness versus the complexity of the representation, with the goal of not unnecessarily increasing the complexity and message payload. The linking between existing healthcare ontologies and the IoT stream ontology will mainly be designed by the stream reasoning results and location-based information of patients and assisting personal. The final representation should enable complex inferences over treatments of an individual patient based on existing patient information and real-time diagnosis results.

5.5.1 Linked data and ontologies review for Healthcare applications

The used ECG data set is available as part of one of the databases at PhysioBank [MM01]. The databases comprehend ECG records containing a continuous recording from a single subject. In MIT-BIH, a signal is defined more restrictively as a finite sequence of integer samples. These are usually obtained by digitalizing a continuous observed function of time at a fixed sampling frequency. All sample intervals for a given signal are equal. MIT DB records are each 30 min in duration and are annotated. This means that a label called an annotation describes each beat. The conceptual model resulting from this standard is depicted in the left-hand of figure 5.18.

This figure illustrates a possible linking of the MIT-BIH dataset with the ECG ontology [GGPF11]. The annotation structure of the MIT-BIH ontology can be of importance in future ML training approaches.

To explicitly describe sensor measurements uniformly, there are proposals to enrich them with semantic web technologies. With an increasing amount of domain-specific IoT applications, the amount of specified domain knowledge-bases is increasing too. Domain applications use ontologies and semantic descriptions as a way to define types, properties, and relationships of entities that exist for a specific domain. In healthcare, many ontologies have been proposed to describe assisted living applica-

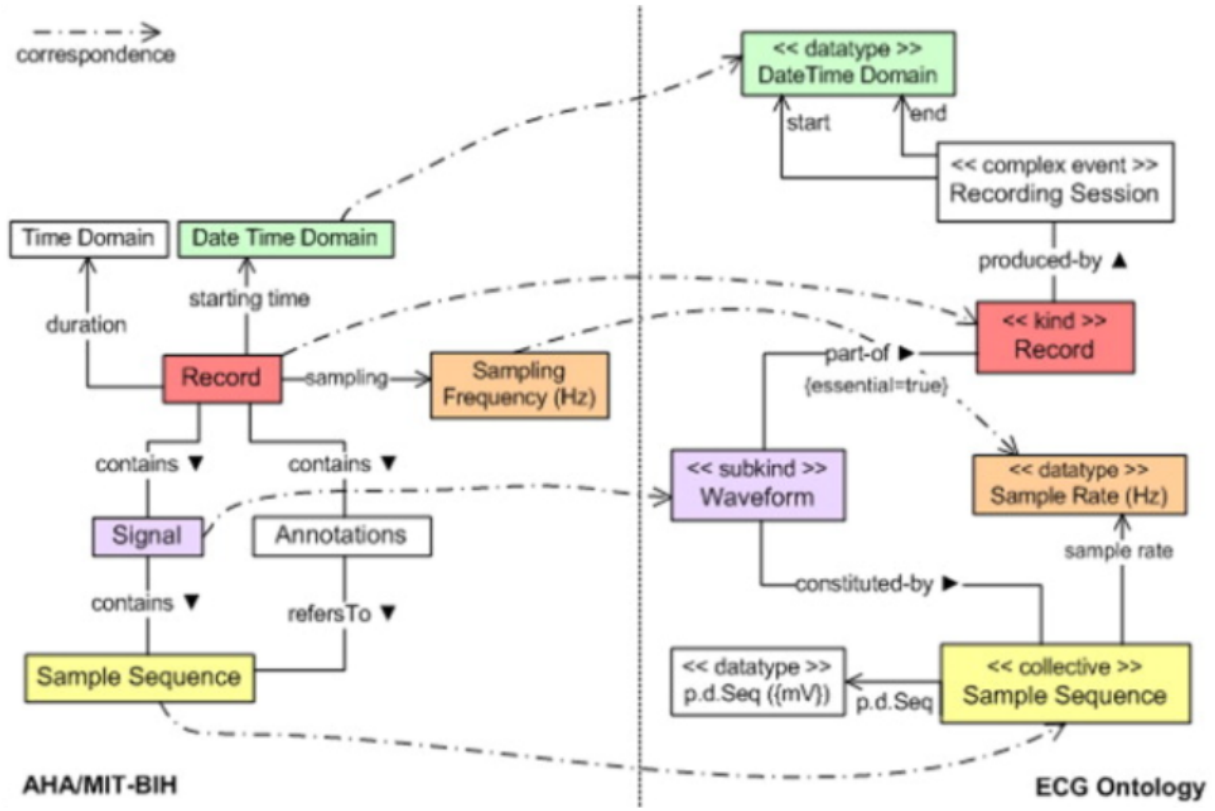

Figure 5.18: PhysioNet ECG data ontology representation [ZCB19]

tions [ZCB19]. The following ontologies have been identified as being of relevance for a cross-domain correlation in this case study:

Medical applications:

1. Disease Ontology (DO): human diseases for linking biomedical knowledge through disease data [KL14].

2. SNOMED-CT: advanced terminology and coding system for eHealth [Don06].

3. OdIH-WS: Ontology-driven Interactive Healthcare with Wearable Sensors[KKDC14]. This ontology has the goal to retrieve in real-time context information in with ontological methods by integrating meteorological data in order to prevent disease.

4. ContoExam [BBS+13] is an ontology developed handle interoperability problems of sensor networks in the context of e-health applications. This ontology contains specific expressions and specifications for medical uses as examination vocabulary and expressions.

5. MetaQ is an ontology-based framework for activity recognition in AAL environments that uses SPARQL queries and OWL 2 activity patterns[MDK15].

AAL applications: There are still issues and challenges in the field of AAL to tackle, such as defining an ontology that describes actions, activities with uncertainty, concurrent, and overlapping activities [HM04].
SSN [CBB+12] is a suitable ontology for sensors description in IoT and can generate data in RDF format. Representing data using RDF or OWL enhances the interoperability between IoT systems [ZCB19]. Several healthcare ontologies have been proposed to describe diseases and activities of daily living.

The European project OpenIoT [KL14] is an open-source middleware platform. OpenIoT represents a on-demand access to cloud-based IoT services for connected objects [ZCB19]. OpenIoT architecture uses the CUPUS middleware as a cloud-based publish and subscribe processing engine for sensing as a service of sensors and relies on SSN sensors description. The stream data is stored as linked data and processed by SPARQL queries. OpenIoT can be viewed as a federation of several interconnected middleware projects in the field of smart cities or the campus and agriculture domain. OpenIoT could have been excellent for IoT health applications, but its complexity due to the variety of middleware solutions can be a significant drawback for the developer [ZCB19].

5.5.1.1 Proposed semantic representation along the data pipeline

For this thesis, the proposed rdf-schemas for the semantic representation of the data can be seen in figure 5.19. The representation followed the minimum requirements of the data processing pipeline and can be extended and mapped to different concepts in the field of healthcare and ECG representations.

Anomaly segment
It has turned out that the definition of an extracted anomlous timeseries of an ECG stream can be hardly defined by many ontologies in healthcare. In this scenario, the SAREF4Health ontology [SAR] offered a solution for a possible representation of this type of data. The proposed representation can be found in figure **??** and contains sequences.

```
Sensor Observations Example of ECG observation
PREF qu: <http://purl.oclc.org/NET/ssnx/qu/qu>
PREF ssn: <http://www.w3.org/ns/ssn/>
PREF ecg: <http://bioportal.bioontology.org/ontologies/1146>
PREF prov: <https://www.w3.org/TR/prov-o/>

:ecg_obs1 a ecg:Observation ;
        ssn:observedProperty ecg:ecg_waveform ;
        ssn:observedBy : ECG_device1 ;
        ssn:observationResult :ecg_obsvalue_1 ;
        time:inXSDDateTime "2019-07-21T16:20:00"^^xsd:datetime .

:ecg_obsvalue_1 a ecg:Sample ;
        qu:numericalValue "345.00"^^xsd:float ;
        qu:unit :millivolt .
```

Figure 5.19: Example of a possible semantic representation of sensor data.

```
@prefix s4h: <https://w3id.org/def/saref4health# > .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns# > .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema# > .
@prefix saref: <https://w3id.org/saref# > .
@prefix saref4envi: <https://w3id.org/def/saref4envi# > .
@prefix sarefInst: <https://w3id.org/saref/instances# > .
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.8/ > .
@prefix owl: <http://www.w3.org/2002/07/owl# > .

Session1_patient100 a s4h::TimeSeriesMeasurements;
      rdfs:label "Time series measurements patient 100";

Anomaly_observation_1 a s4h::ECGSampleSequence;
      rdfs:subClassOf: Session1_patient100 ;
      saref:isMeasuredIn = ElectricPotential_MilliVolts;
      saref:relatesToProperty(): only HeartElectricalActivity;
      saref:hasTimestap= "2018-04-22T22:15:30"^^xsd:dateTime ;

Anomaly_segment_1 asarefinst::ECGMeasurementsSeries_Example;
      rdf:type: Anomaly_observation_1  ;
      rdfs:label "sequence 001 -Patient 100 - ECG measurements series " ;
      saref4envi:hasFrequencyMeasurement sarefInst:FrequencyOf256Hertz ;
      saref:hasTimestamp= "2018-04-22T22:15:30"^^xsd:dateTime ;
      saref:hasValues =1002,1432,1222 (...);
```

Figure 5.20: Example of a possible semantic representation of anomalous segment.

Analytics data set descriptions

The ECG recording produces a data flow in which intermediate datasets can potentially be reused for other purposes, including other analytic methods, or data processing evaluations. The produced datasets in the data pipeline can be systematically annotated, starting from the raw datasets produced by the IoT ECG device, the anomaly detection process, classification as well as a possible complex event processing.

```
Monitoring Result
:classified_events_patient100_2019-07-21 a prov:Entity;
        prov:wasGeneratedBy :physiologicalModel_patient1_2019-07-21;
        prov:wasDerivedFrom :ecg_patient1_2019-07-21;
        prov:wasAttributedTo :1dCNN.

:1dCNN a prov:Agent;
        prov:actedOnBehalfOf: doctorX .

:physiologicalModel_patient100_2019-07-21 a prov:Activity;
        prov:used :ecg_patient100_2019-07-21;
        prov:wasAssociatedWith :1dCNN;
        prov:endedAtTime "2019-07-21T17:02:02Z"^^xsd:dateTime.
```

Figure 5.21: Example of a possible semantic representation of derived event.

The monitoring result shows how the high level classified events of the CNN model is annotated as a derived dataset from the processed sensor data. The PROV-O7 ontology is used, as recommended by W3C when interchanging and representing provenance information [W3C] . It contains the following main concepts:

- Entities: Physical or digital entities.
- Activities: Actions that are performed on an entity like transformation or processing.
- Agents: perform or are responsible for the activities upon the entities.

The example in the figure 5.21 represents a classified event dataset as an entity, which is generated by an activity, described as a physiological model performed by the CNN algorithm.

5.5.1.2 Conclusion and ongoing challenges in the context of semantic enrichment

Some projects have proposed to tackle the interoperability issue from a technological perspective by relying on middleware solutions that promote the interoperability and the manageability of sensors in an IoT system. Although the existing semantic middleware proposals address many challenges and requirements regarding the interoperability in IoT systems, there are still open research challenges related to scalability and real-time reasoning. Using ontologies affects the requirements as parsing, storing, inferencing, and querying over RDF data takes a longer time compared to simple

data formats. Furthermore, do ontologies require domain knowledge and expertise and require a higher computational cost. The complexity is considered to be related to the diversity of libraries to use and the complexity of the programming environment [ZCB19]. Another challenge that could be addressed in the research area is related to ontologies for sensors and domain descriptions. Providing a complete ontology that combines the health care domain and sensors in IoT is still an ongoing challenge. Using *Message oriented Middleware* (MOM) approaches with semantic descriptions in IoT is still in an early stage.

Web of Thing (WoT) [GTMW11] can be seen as a major solutions for interoperability issues in IoT. It allows an easier way for IoT applications to build upon smart things. The WoT concept relies on the connectivity service of IoT and easy access to sensors data[GT09]. It can be seen as an evolution of the Internet of things where all components share their information and collaborate to generate and infer advanced knowledge. Using semantic descriptions enables data contextualization for data stream discovery and querying. New research shifts to the semantic web of things [PRB+11] where data can be integrated with data and available services. WoT is a open research to improve and investigate several IoT environments, including healthcare.

Chapter 6

Conclusion and future work

This chapter contains the conclusion and future work of the general review of the state of the art of the three research questions in the field of IoT, as well as the case study related conclusion, critical assessment, and future work.

6.1 Conclusion

General: The review explored the state of the art in the field of IoT stream processing, semantic enrichment, and complex event processing. Current research focuses heavily on the application of modern machine learning algorithms in nearly every use case scenario. While the field of stream processing is widely covered by industry and academia, the fields of domain ontologies show the potential to grow in terms of domain-specific ontologies for IoT applications. The enabling of semantic representation of IoT data will be required for semantic complex event processing scenarios. The motivation for a focus on defining domain-specific IoT domains and semantic complex event processing should be based on the growing interconnection of devices as well as the new resulting possibilities for higher-order reasoning. While in recent years, a lack of SCEP engines for multiple heterogeneous streams represented a significant problem, we now have engines like SPAseq that extend SPARQL with new SCEP operators that can be used over RDF graph-based events that offer new opportunities in the field of complex IoT reasoning. The main contribution of this thesis is the use case independent review of the foundation for an end-to-end IoT ecosystem.

The second contribution consists of a use case based prototype that enables an end-to-end data pipeline process with the requirements for a generalized and scalable

IoT stream ecosystem in the field of eHealth. The implementation can be regarded as a basis for other use cases as a fast prototype proposal for small teams with limited expertise. With python being a popular language in the rapidly expanding field of data science, an entry barrier was kept low by realizing the ecosystem proposal solely based on Python and SQL.

Case Study: The presented use case of ECG-data classification illustrates a scalable distributed processing scenario, where anomalous segments are detected on the edge device with the HTM algorithm to reduce the data load on the architecture. The Kafka system stores the identified abnormal segments. Single signals were successfully extracted from the segment by a Faust worker node, which analyzed it with a trained CNN model on possible heart conditions. The HTM algorithm was tested on a raw and noisy test set and was able to reduce the stream volume by up to 40% while keeping a TPR of detected unhealthy signals at 94%. The total throughput time, starting from sensing until the classified event, took between 0.5 and 2.2 seconds. Classified events are stored in the Kafka system, where a CEP consumer was attached to reason over the Kafka message and event topics. In this work, Kafka Streams and its higher-level abstraction language KSQL was deployed to analyze the proof of concept of CEP integration over a stress-sensor stream and the heart rate sensor, where a positive composite event detection led to an alarm. This scenario can be considered as a trigger to inform caregivers with the diagnosis results and recommended actions and treatments. With the foundation of CEP, further research was based on enabling high inference options based on semantic integration of stream data, historical patient information, and healthcare knowledge bases. ECG ontologies turned out to be not satisfying for the IoT use case example, due to missing expressivity for segment and stream data. The SAREF4health and SSN ontology, in combination with healthcare ontologies, have been identified for the representation of the ECG data streams in the scenario of segment processing. The system foundation for a patient-specific real-time prescriptive treatment is given in the context of this architecture.

6.2 Critical assessment

Case Study: While the HTM algorithm can be used to reduce the load on the classification component inside the Fog or central analytics layer, CNN models are already able to run in IoT devices itself. Depending on the scenario, considerations for the positioning of pre-processing, signal extract, and classification need to be made. While the unoptimized HTM algorithm showed promising results on a raw and noisy data stream, an optimized HTM implementation should be evaluated against state of the art algorithms on pre-processed streams. Due to the limited power of edge devices, another evaluation criterion should be the resource requirements for running the HTM algorithm against other anomaly detection algorithm. In different scenarios, the device resource restrictions or the goal of saving battery power could limit the practicality of a more sophisticated algorithm like HTM.

Another use case specific consideration is the open research field of Edge distributed CEP. By reasoning over multiple streams right at the source of creation, privacy issues and data overloading can be prevented, as well as latency decreased.

6.3 Future Work

General: Semantic complex event processing is the product of the synergy of those three domains and represents a challenge, where domain experts and developers need to work closer together. The development in modern IoT architecture concepts should be followed carefully, where concepts like misc computing and decentralized resource offloading will enable new opportunities as well as challenges to stream and complex event processing, as well as semantic enrichment on the Edge or Fog layer.

Current research in the field of IoT has a strong focus on leveraging the computing power of edge or fog systems. A future focus should be in the area of fog-centric architecture designs. A fully-autonomous workload distribution mechanism is already presented with examples like EdgeCEP [CYH+17]. An exciting project to follow is the Apache Edgent incubator project, which is a programming model embedded in gateways and small edge devices, enabling local, real-time analytics on continuous data streams [APA].

Case Study: The field of data privacy was not in the scope of this thesis but should be considered as one of the most critical requirements of an eHealth IoT stream application. For the storage and communicate of eHealth data, a robust encryption mechanism needs to be implemented. [XLZ+14] proposes a physician authorization mechanism, where the data is sent encrypted to the cloud and is made accessible on the approval of patients by a doctor. With wearable monitoring devices, such data can be encrypted and stored online, where the patient allows physicians to access and reason over the already gathered insights.

The case study has been mainly focused on the end-to-end integration, where the optimization of the anomaly detection and the classification algorithm where tasks outside the scope of the thesis. The HTM algorithm can be improved with hyperparameter optimization as well as pre-processing steps to clean the raw ECG stream. The evaluation of the pre-processing should contain considerations for dimensionality reduction and representation techniques to improve anomaly detection sensitivity, latency, and resource utilization. Another task would be to replace the 1D-CNN model with a state of the art 2D-CNN model. The model of Jun et al. [JNK+18] can be recreated and exchanged with the current model, without the need for changing the processing pipeline, due to the same approach of signal extraction. The signal will be transformed into greyscale images, where the deployment can be expanded in even analyzing the patient's ECG monitor through the camera.

While the proposed ecosystem design offers a generalized IoT ecosystem for consumer-based SCEP components, a real use case should be determined with experts in the medical domain to find a benchmark between traditional medical analytic scenarios and event-based real-time analytics. A synergy between healthcare expert systems and reliable real-time analytics offers a wide range of new use cases to improve the life of patients.

Chapter 7

Appedix

7.1 HTM description

Step 1 - Create an SDR of the input by activating whole columns

The first step determines the active columns of cells in HTM (see Figure 7.1). Each column is connected to a subset of the input bits via the synapses on a proximal dendrite. Subsets for different columns may overlap but they are not equal. Consequently, different input patterns result in different levels of activation. The columns with the strongest activation level block columns with weaker activation. The size of the inhibition area around a column is adjustable and can span from very small to the entire region. The inhibition mechanism ensures a sparse representation of the input. If only a few input bits change, some columns will receive a few more or a few less active one inputs, but the set of active columns is not likely to change much. Therefore, similar input patterns will map to a relatively stable set of active columns.

HTM learns by forming and reforming connections between cells. Learning occurs by updating the permanence values of the synapses. Only the active columns increment the permanence value of synapses connected to active bits and decrement otherwise. Columns that do not become active for a long period do not learning anything. To not waste columns, the overlap scores of these columns are "boosted" to ensure that all columns partake in the learning of patterns.

Step 2 - Place the input in context by selecting among cells in active columns. The cells can be in one of three different states. If a cell is active because of a feed-forward input, it gets assigned the active state. If the cell is active because of lateral connections with nearby cells, then it is in the predictive state and otherwise in the

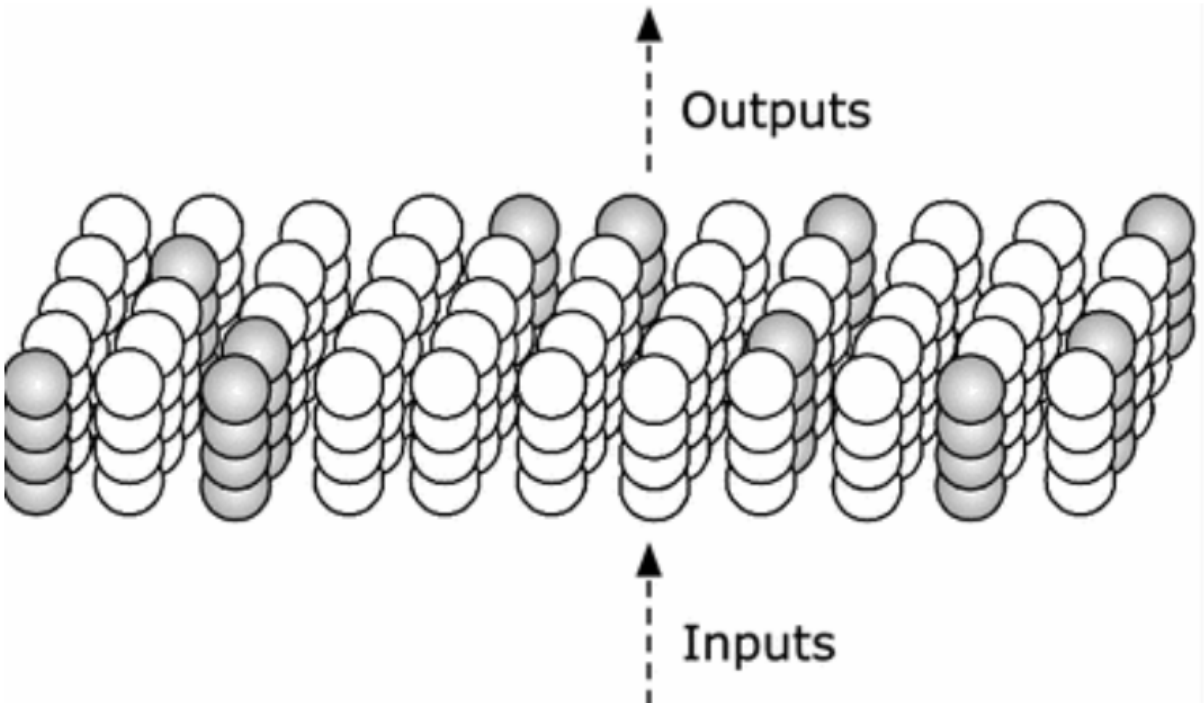

Figure 7.1: HTM SDR cell column structure [Hol16].

inactive state.

The second step converts the column representation of the input into a new one that includes the past context. This new representation is formed by the activation of a cells subset within each column (normally only one cell per column). The rule used to activate cells is as follows: When a column becomes active, HTM checks all the cells in the column. When one or several cells inside the column are in the predictive state, only those cells get the state active. If there are no cells in the column in the predictive state, then all cells become active. This means, if an input pattern is expected, then HTM approves the pattern by activating solely the cells in the predictive state. If an unexpected input pattern follows, then HTM activates all the cells in the column.

HTM can represent the exact same input different in another context, by selecting different active cells in each active column. Figure 7.2 illustrates how HTM can represent the sequence AB as part of two larger sequences CABF and HABG. The same columns have active cells in both cases but the active cells differ. If there is no prior state and therefore no prediction, the cells in a column will be activated when the column gets set on active.

Step 3 - Predict future patterns from learned transitions between SDRs The third and final step makes a prediction of likely new input. The prediction is based on the representation formed in the second step, which includes context from all previ-

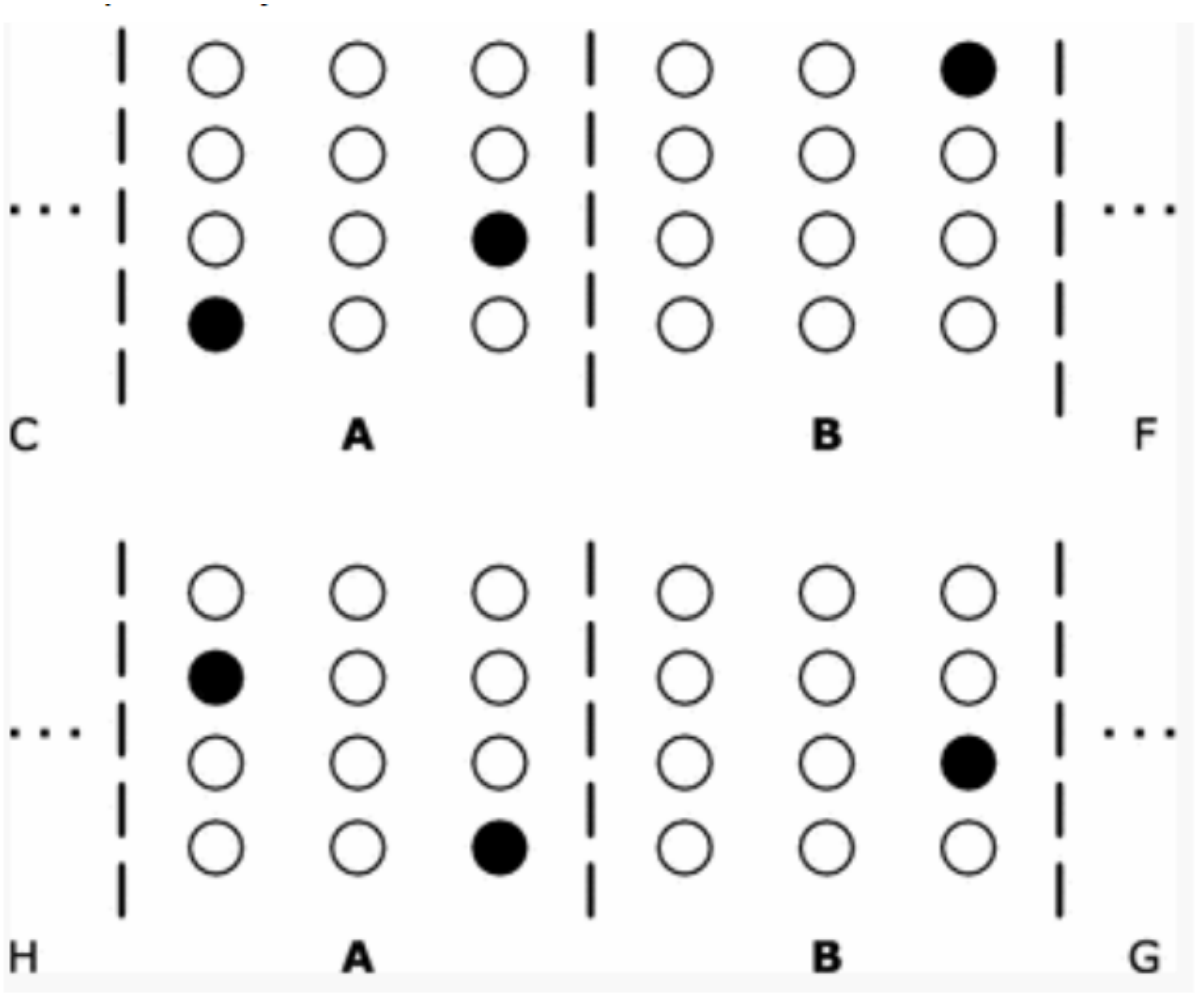

Figure 7.2: Representation of specific sequences in larger sequences [Hol16].

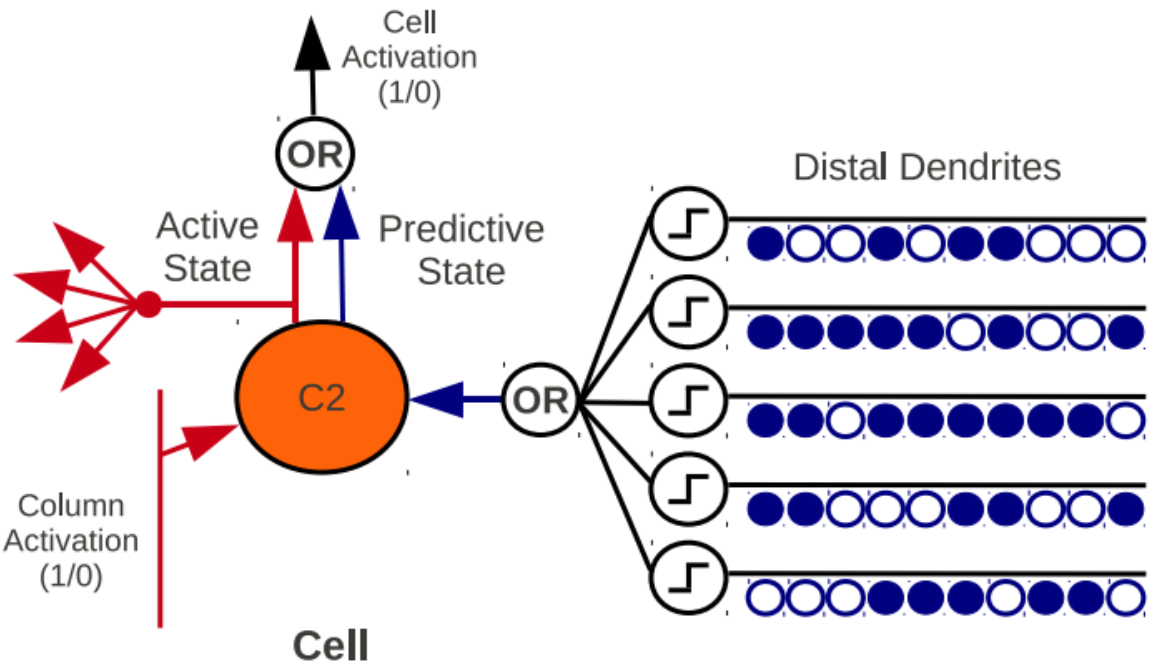

Figure 7.3: Distal segment representation[Pri11].

ous input patterns. When HTM makes a prediction, all cells that are likely to become active due to future feed-forward input are changed to the predictive state. Because the sparse representations, multiple predictions can be made at the same moment. The cells in the predictive state represent the HTM prediction for the next input.

The predictive state of any cell in HTM is determined by its distal segments. A segment connects to cells via synapses on distal dendrites. If enough of these cells are active, then the segment becomes active (see Figure 7.3). A cell switches to the predictive state when it has at least one active segment. However, a cell that is already active from the second step does not switch to the predictive state. Learning occurs by adjusting the permanence values of the synapses on active segments at every time step. The permanence of a synapse is only updated when a predicted cell actually becomes active during the next time instance. The permanence of a synapse connecting to an active cell is increased while the permanence of a synapse to an inactive cell is decreased.

7.2 ECG anomaly with HTM code

Input: incoming-data-t(ecg-signals as unbounded stream in time-step t), flag-send= anomaly based trigger for sending data, Queue-size = size of data queue, segment-size= size of anomalous segment for further analysis Output: Anomalous segment for further processing

initialize

```
data_queue(size=queue_size)     #FIFO Queue
moving_average_queue(size=segment_size) #FIFO queue

while device_is_online do:
    data_queue(insert=incoming_data_t)
    anomaly_score_t=HTM_anomaly_test(incoming_data_t)
    moving_average_queue(insert=anomaly_score_t)
    if moving_average_queue>=threshold
        flag_send=True
    if flag_send==true && length(data_queue)>=segment_size
        send_to_mqtt_broker(data_queue_dequeue(size=segment_size)
        falg_send=False
```

7.3 Signal extract

Input: is (Input segment received by anomaly detection)
Output: Signals (n signals extracted of input-segment **initialize** previous=0

```
While worker online:
is_norm=(is_s-min(is_s))/(max(is_s)-min(is_s))
is_grad=gradient(is_norm)
        zero_crossing=zero_crossing(is_grad)
        peaks_index=is_norm[zero_crossing]>=0.9
        for peak in peaks_index
            peak_distances<-(peak-previous)
            previous=peak
```

```
median_length=median(peak_distances)
segment_length=median_length*1.2
for signal in peaks_distances
    signal=is_norm[signal_index:(signal + median_length)]
    signal=fillup_zeros(signal, segment_length)
```

Bibliography

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, Jun 2006.

[AFRS11] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: A unified language for event processing and stream reasoning. pages 635–644, 01 2011.

[APA] Apache software foundation - apache edgent. `https://edgent.apache.org/`. Accessed: 2019-09-02.

[ARFS12] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in etalis. *Semantic Web*, 3:397–407, 2012.

[aut] Automotive council uk. `https://www.automotivecouncil.co.uk.2018`. Accessed: 2019-08-20.

[Bag13] Anthony Bagnall. A run length transformation for discriminating between auto regressive time series. *Journal of Classification*, 31, 07 2013.

[BBC+09] Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql: Sparql for continuous querying. 01 2009.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16, 2002.

[BBDR13] S. Beckstein, R. Bruns, Jürgen Dunkel, and L. Renners. Integrating se-

mantic knowledge in data stream processing. *CEUR Workshop Proceedings*, 1070:1–12, 01 2013.

[BBS+13] Paul Brandt, Twan Basten, Sander Stuiik, Vinh Bui, Paul de Clercq, Luís Ferreira Pires, and Marten van Sinderen. Semantic interoperability in sensor applications making sense of sensor data. In *2013 IEEE Symposium on Computational Intelligence in Healthcare and e-health (CICARE)*, pages 34–41. IEEE, 2013.

[BC19] D Berndt and J Clifford. Finding patterns in time series in advances. 05 2019.

[BD16] Rajkumar Buyya and A.V. Dastjerdi. *Internet of Things: Principles and Paradigms*. 05 2016.

[BGHS12] Payam Barnaghi, Frieder Ganz, Cory Henson, and Amit Sheth. Computing perception from sensor data. pages 1–4, 10 2012.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[BOE17] A. C. Baktir, A. Ozgovde, and C. Ersoy. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Communications Surveys Tutorials*, 19(4):2359–2391, Fourthquarter 2017.

[BRK+05] Anthony J. Bagnall, Chotirat Ratanamahatana, Eamonn J. Keogh, Stefano Lonardi, and Gareth J. Janacek. A bit level representation for time series data mining with shape based similarity. *Data Mining and Knowledge Discovery*, 13:11–40, 2005.

[BWHT12] Payam Barnaghi, Wei Wang, Cory Henson, and KERRY TAYLOR. Semantics for the internet of things: Early progress and back to the future. *International Journal on Semantic Web Information Systems*, 8, 01 2012.

[C16] Whitepaper: Cisco internet of things at a glance 2016. Accessed: 2019-09-14.

[CBB+12] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, David Kelsey, Danh Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, and Kerry Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17, 12 2012.

[CCG10] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web – ISWC 2010*, pages 96–111, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CM12a] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44, 06 2012.

[CM12b] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.

[CM12c] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44, 06 2012.

[CMC] Mustafa S Cetin, Abdullah Mueen, and Vince D. Calhoun. *Shapelet Ensemble for Multi-dimensional Time Series*, pages 307–315.

[CMZ07] Graham Cormode, Senthilmurugan Muthukrishnan, and Wei Zhuang. Conquering the divide: Continuous clustering of distributed data streams. pages 1036–1045, 05 2007.

[CP08] Marcella Corduas and Domenico Piccolo. Time series clustering and classification by the autoregressive metric. *Computational Statistics Data Analysis*, 52:1860–1872, 02 2008.

[CYH+17] S. Choochotkaew, H. Yamaguchi, T. Higashino, M. Shibuya, and T. Hasegawa. Edgecep: Fully-distributed complex event processing on

iot edges. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 121–129, June 2017.

[DC18] Lipika Deka and Mashrur Chowdhury. *Transportation Cyber-Physical Systems*. Elsevier, 2018.

[DD17] K. Dolui and S. K. Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2017.

[DK12] Anusuriya Devaraju and Tomi Kauppinen. Sensors tell more than they sense: Modeling and reasoning about sensor observations for understanding weather events. *Special Issue on Semantic Sensor Networks, International Journal of Sensors, Wireless Communications and Control*, 2, 10 2012.

[Don06] Kevin Donnelly. Snomed-ct: The advanced terminology and coding system for ehealth. *Studies in health technology and informatics*, 121:279, 2006.

[DRTM13] Houtao Deng, George C. Runger, Eugene Tuv, and Vladimir Martyanov. A time series forest for classification and feature extraction. *Inf. Sci.*, 239:142–153, 2013.

[DTLNW13] Hoang Dinh Thai, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13, 12 2013.

[EHPdAS16] Markus Endler, Edward Haeusler, Vitor Pinheiro de Almeida, and Francisco Silva. Towards real-time semantic reasoning for the internet of things, 10 2016.

[Eva11] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. 2011.

[FPY+16] Ruogu Fang, Samira Pouyanfar, Yimin Yang, Shu-Ching Chen, and SS Iyengar. Computational health informatics in the big data age: a survey. *ACM Computing Surveys (CSUR)*, 49(1):12, 2016.

[GAB+16] A. Gyrard, G. Atemezing, C. Bonnet, K. Boudaoud, and M. Serrano. Reusing and unifying background knowledge for internet of things with

lov4iot. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 262–269, Aug 2016.

[GBB14] Amelie Gyrard, Christian Bonnet, and Karima Boudaoud. Demo paper: Helping iot application developers with sensor-based linked open rules. *CEUR Workshop Proceedings*, 1401:105–108, 01 2014.

[GBBS16] A. Gyrard, C. Bonnet, K. Boudaoud, and M. Serrano. Lov4iot: A second life for ontology-based domain knowledge to build semantic web of things applications. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 254–261, Aug 2016.

[GBC13] Frieder Ganz, Payam Barnaghi, and Francois Carrez. Information abstraction for heterogeneous real world internet data. *Sensors Journal, IEEE*, 13, 10 2013.

[GBC14] Frieder Ganz, Payam Barnaghi, and Francois Carrez. Automated semantic knowledge acquisition from sensor data. *IEEE Systems Journal*, 10:1–12, 08 2014.

[GGPF11] Bernardo Gonçalves, Giancarlo Guizzardi, and José Pereira Filho. Using an ecg reference ontology for semantic interoperability of ecg data. *Journal of biomedical informatics*, 44:126–36, 02 2011.

[GKK+19a] M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Dustdar, O. Scekic, T. Rausch, S. Nastic, S. Ristov, and T. Fahringer. A deviceless edge computing approach for streaming iot applications. *IEEE Internet Computing*, 23(1):37–45, Jan 2019.

[GKK+19b] Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Schahram Dustdar, Ognjen Scekic, Thomas Rausch, Stefan Nastic, Sasko Ristov, and Thomas Fahringer. A deviceless edge computing approach for streaming iot applications. *IEEE Internet Computing*, 23:37–45, 2019.

[GMM18a] Federico Giannoni, Marco Mancini, and Federico Marinelli. Anomaly detection models for iot time series data, 11 2018.

[GMM18b] Federico Giannoni, Marco Mancini, and Federico Marinelli. Anomaly detection models for iot time series data, 11 2018.

[GT09] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, volume 15, 2009.

[GTMW11] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of things*, pages 97–129. Springer, 2011.

[GZPL18] Syed Gillani, Antoine Zimmermann, Gauthier Picard, and Frederique Laforest. A query language for semantic complex event processing: Syntax, semantics and implementation. *Semantic Web*, 10:1–41, 08 2018.

[HBZZ17] Pengzhan Hao, Yongshu Bai, Xin Zhang, and Yifan Zhang. Edgecourier: an edge-hosted personal service for low-bandwidth document synchronization in mobile cloud storage services. pages 1–14, 10 2017.

[HGBV00] Jean-Pierre Hubaux, Th. W. Gross, J.-Y. Le Boudec, and Martin Vetterli. Towards self-organized mobile ad hoc networks : the terminodes project. 2000.

[HLB+14] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.*, 28(4):851–881, July 2014.

[HM04] Markus C Huebscher and Julie A McCann. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 111–116. ACM, 2004.

[Hol16] Kjell Jørgen Hole. *The HTM Learning Algorithm*, pages 113–124. Springer International Publishing, Cham, 2016.

[HPX11] Sven Helmer, Alexandra Poulovassilis, and Fatos Xhafa. *Reasoning in Event-Based Distributed Systems*, volume 347. 01 2011.

[HQG+12] Junxian Huang, Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *MobiSys*, 2012.

[HST12] Cory Henson, Amit Sheth, and Krishnaprasad Thirunarayan. Semantic perception: Converting sensory observations to abstractions. 16:26–34, 03/2012 2012.

[Hud16] Mahesh Huddar. Iot streaming applications and challenges to achieve real-time qos. *Journal of Advances in Science and Technology*, 12:65–69, 01 2016.

[HYX+16] Xiaohui Huang, Yunming Ye, Liyan Xiong, Raymond Lau, Nan Jiang, and Shaokai Wang. Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences*, 367, 06 2016.

[IKK+15] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak. The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[JKJO11] Youngseon Jeong, Myong Kee Jeong, and Olufemi Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44:2231–2240, 09 2011.

[JNIH16] Michael Jones, Daniel Nikovski, Makoto Imamura, and Takahisa Hirata. Exemplar learning for extremely efficient anomaly detection in real-valued time series. *Data Mining and Knowledge Discovery*, 30:1–28, 01 2016.

[JNK+18] Tae Joon Jun, Minh Hoang Nguyen, Daeyoun Kang, Dohyeun Kim, Daeyoung Kim, and Young-Hak Kim. Ecg arrhythmia classification using a 2-d convolutional neural network. *ArXiv*, abs/1804.06812, 2018.

[Kafa] Kafka 2.3 documentation. `https://kafka.apache.org/documentation/`. Accessed: 2019-09-05.

[Kafb] Kafka streams. `https://kafka.apache.org/documentation/streams/`. Accessed: 2019-09-05.

[KJK11] Ankesh Khandelwal, Ian Jacobi, and Lalana Kagal. Linked rules: Principles for rule reuse on the web. pages 108–123, 08 2011.

[KKDC14] Jonghun Kim, Jae-Kwon Kim, Lee Daesung, and Kyung-Yong Chung. Ontology driven interactive healthcare with wearable sensors. *Multimedia Tools and Applications*, 71, 07 2014.

[KL14] J. Kim and J. Lee. Openiot: An open service framework for the internet of things. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 89–93, March 2014.

[KNL+18] Farah Karim, Ola Al Naameh, Ioanna Lytra, Christian Mader, Maria-Esther Vidal, and Sören Auer. *Semantic Enrichment of IoT Stream Data On-demand*. IEEE, Laguna Hills, CA, USA, 2018.

[KP00] Eamonn J. Keogh and Michael J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 285–289, New York, NY, USA, 2000. ACM.

[KR04] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping, 2004.

[LCL16] Kathrin Rodriguez Llanes, Marco A. Casanova, and Noel Moreno Lemus. From sensor data streams to linked streaming data: a survey of main approaches. *JIDM*, 7:130–140, 2016.

[LCW07] Xiang Lian, Lei Chen, and Bin Wang. Approximate similarity search over multiple stream time series. pages 962–968, 04 2007.

[LDS15] Boduo Li, Yanlei Diao, and Prashant Shenoy. Supporting scalable analytics with latency constraints. *Proceedings of the VLDB Endowment*, 8:1166–1177, 07 2015.

[LJGC+17] Hemank Lamba, Thomas J. Glazier, Javier Cámara, Bradley Schmerl, David Garlan, and Juergen Pfeffer. Model-based cluster analysis for identifying suspicious activity sequences in software. pages 17–22, 03 2017.

[LKL12] Jessica Lin, Rohan Khade, and Yuan Li. K rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39, 10 2012.

[LPK06] Seung-Hwan Lim, Heejin Park, and Sang-Wook Kim. Using multiple indexes for efficient subsequence matching in time-series databases. volume 177, pages 65–79, 01 2006.

[MBSRJ18] Karl Øyvind Mikalsen, Filippo Maria Bianchi, Cristina Soguero-Ruíz, and Robert Jenssen. Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, 76:569–581, 2018.

[MC11] Alessandro Margara and Gianpaolo Cugola. Processing flows of information: from data stream to complex event processing. pages 359–360, 01 2011.

[MDK15] Georgios Meditskos, Stamatia Dasiopoulou, and Ioannis Kompatsiaris. Metaq: A knowledge-driven framework for context-aware activity recognition combining sparql and owl 2 activity patterns. *Pervasive and Mobile Computing*, 5, 02 2015.

[MGT+17] Higinio Mora, David Gil, Rafael Muñoz Terol, Jorge Azorín, and Julian Szymanski. An iot-based computational framework for healthcare monitoring in mobile environments. *Sensors*, 17(10):2302, 2017.

[MIT] Mit-bih arrhythmia database directory. `https://archive.physionet.org/physiobank/database/html/mitdbdir/mitdbdir.htm`. Accessed: 2019-09-05.

[MM01] G.B. Moody and R.G. Mark. The impact of the mit-bih arrhythmia database. *IEEE engineering in medicine and biology magazine : the quarterly magazine of the Engineering in Medicine Biology Society*, 20:45–50, 06 2001.

[MM12] A. Moraru and D. Mladenić. A framework for semantic enrichment of sensor data. In *Proceedings of the ITI 2012 34th International Conference on Information Technology Interfaces*, pages 155–160, June 2012.

[MMV+11] Alexandra Moraru, Dunja Mladenić, Matevz Vucnik, Maria Porcius, Carolina Fortuna, and Mihael Mohorcic. Exposing real world information for the web of things. page 6, 03 2011.

[MNG+17] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[Mor13] Andrei Galbeaza Moraru. Enrichment of sensor descriptions and measurements using semantic technologies. 2013.

[Mou78] Vernon Mountcastle. An organizing principle for cerebral function: the unit module and the distributed system. *The mindful brain*, 1978.

[MPGS15] Albert Meroño-Peñuela, Christophe Guéret, and Stefan Schlobach. Linked edit rules: A web friendly way of checking quality of rdf data cubes. In *SemStats@ISWC*, 2015.

[MPP+10] Alexandra Moraru, Marko Pesko, Maria Porcius, Carolina Fortuna, and Dunja Mladenić. Using machine learning on sensor data. *CIT*, 18, 01 2010.

[MPSRS+17] Pedro M. Pinto Silva, Joao Rodrigues, Joaquim Silva, Rolando Martins, Luis Lopes, and Fernando Silva. Using edge-clouds to reduce load on traditional wifi infrastructures and improve quality of experience. pages 61–67, 05 2017.

[MQT] Mqtt org website. `http://mqtt.org/`. Accessed: 2019-09-05.

[MTL+16] Carrie MacGillivray, Vernon Turner, Lionel Lamy, Kevin Prouty, Rebecca Segal, Andrea Siviero, Marcus Torchia, Dan Vesset, Robert Westervelt, and Ruthbea Yesner. *IDC FutureScape: Worldwide Internet of Things 2017 Predictions*. IDC, 2016.

[MTZ16a] Raef Mousheimish, Yehia Taher, and Karine Zeitouni. Automatic learning of predictive rules for complex event processing: doctoral symposium. pages 414–417, 06 2016.

[MTZ16b] Raef Mousheimish, Yehia Taher, and Karine Zeitouni. Complex event processing for the non-expert with autocep: demo. In *DEBS*, 2016.

[MWY+17] Y. Ma, Y. Wang, J. Yang, Y. Miao, and W. Li. Big health application system based on health internet of things and big data. *IEEE Access*, 5:7885–7897, 2017.

[Nis19] Tahir M. Nisar. *Smartphone and App Implementations That Improve Productivity*. De—G Press, 2019.

[PDTPH11] Danh Le Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *International Semantic Web Conference*, 2011.

[PRB+11] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, et al. Spitfire: Toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, 2011.

[Pri11] Ryan Price. Hierarchical temporal memory cortical learning algorithm for pattern recognition on multi-core architectures. 2011.

[PTJ+15] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, July 2015.

[RAM05] Juan José Rodríguez, Carlos J. Alonso, and José A. Maestro. Support vector machines of interval-based features for time series classification. *Know.-Based Syst.*, 18(4-5):171–178, August 2005.

[RGSE14] Fano Ramparany, Fermín Galán, Javier Soriano, and Tarek Elsaleh. Handling smart environment devices, data and services at the semantic level with the fi-ware core platform. 10 2014.

[Roc] Rocksdb website. `https://rocksdb.org/`. Accessed: 2019-09-05.

[RP17] B. Ravandi and I. Papapanagiotou. A self-learning scheduling in cloud software defined block storage. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 415–422, June 2017.

[RZZS15] J. Ren, Y. Zhang, K. Zhang, and X. Shen. Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. *IEEE Communications Magazine*, 53(3):98–105, March 2015.

[SA15] Joan Serrà and Josep Lluís Arcos. Particle swarm optimization for time series motif discovery. *Knowledge-Based Systems*, 92, 01 2015.

[SAGB14] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *IEEE Communications Surveys Tutorials*, 16(1):369–392, First 2014.

[SAR] Saref4health ontology. `https://w3id.org/def/saref4health`. Accessed: 2019-08-24.

[Sat96] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 1–7, New York, NY, USA, 1996. ACM.

[SBCD09] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, October 2009.

[SG17] Marcos Serrano and Amelie Gyrard. 6 a review of tools for iot semantics and data streaming analytics. 2017.

[SGKB13] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *IEEE Communications Surveys Tutorials*, 15(3):1294–1313, Third 2013.

[SHSS08] Amit Sheth, Cory Henson, and Satya S. Sahoo. Semantic sensor web. *Internet Computing, IEEE*, 12:78–83, 08 2008.

[SHT16] Maninder Pal Singh, Mohammad Ashraful Hoque, and Sasu Tarkoma. Analysis of systems to process massive data stream. *CoRR*, abs/1605.09021, 2016.

[SN16] Ahmed Salem and Tamer Nadeem. Lamen: leveraging resources on anonymous mobile edge nodes. In *S3@MobiCom*, 2016.

[SPC+14] Agusti Solanas, Constantinos Patsakis, Mauro Conti, Ioannis S Vlachos, Victoria Ramos, Francisco Falcone, Octavian Postolache, Pablo A Pérez-Martínez, Roberto Di Pietro, Despina N Perrea, et al. Smart health: a context-aware health paradigm within smart cities. *IEEE Communications Magazine*, 52(8):74–81, 2014.

[W3C] W3c prov-overview. https://www.w3.org/TR/prov-overview/. Accessed: 2019-09-08.

[W3C13] W3C. Sparql 1.1 overview. https://www.w3.org/TR/sparql11-overview/, 2013. Accessed: 2019-03-26.

[W3C14] W3C. Rdf 1.1 primer. https://www.w3.org/TR/rdf11-primer/, 2014. Accessed: 2019-03-26.

[W3C16] W3C. Rdf stream processing: Requirements and design principles. http://streamreasoning.github.io/RSP-QL/RSP_Requirements_Design_Document/#count-based-window-based-on-basic-graph-pattern, 2016. Accessed: 2019-03-26.

[WB09] Wang Wei and Payam Barnaghi. Semantic annotation and reasoning for sensor data. In Payam Barnaghi, Klaus Moessner, Mirko Presser, and Stefan Meissner, editors, *Smart Sensing and Context*, pages 66–76, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[WHO] World health organisation cardiovascular diseases (cvds). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). Accessed: 2019-09-02.

[wik] Wikimedia commons - sinusrhythmlabels. Accessed: 2019-09-05.

[XAB+11] F. Xhafa, E. Asimakopoulou, N. Bessis, L. Barolli, and M. Takizawa. An event-based approach to supporting team coordination and decision making in disaster management scenarios. In *2011 Third International Conference on Intelligent Networking and Collaborative Systems*, pages 741–745, Nov 2011.

[XGP+15] Xiaoning Xu, Chuancong Gao, Jian Pei, Ke Wang, and Abdullah Al-Barakati. Continuous similarity search for evolving queries. *Knowledge and Information Systems*, 10 2015.

[XLZ+14] Fatos Xhafa, Jingwei Li, Gansen Zhao, Jin Li, Xiaofeng Chen, and Duncan Wong. Designing cloud-based electronic health record system with attribute-based encryption. *Multimedia Tools and Applications*, 74, 05 2014.

[YFN+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna P. Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *ArXiv*, abs/1808.05283, 2019.

[YJK09] Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a new primitive for data mining. pages 947–956, 06 2009.

[YKR08] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. *Knowledge and Information Systems*, 17:241–262, 11 2008.

[ZCB19] Rita Zgheib, Emmanuel Conchon, and Rémi Bastide. *Semantic Middleware Architectures for IoT Healthcare Applications*, pages 263–294. Springer International Publishing, Cham, 2019.

[ZMK+15] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward A. Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *HotCloud*, 2015.